



**João Gabriel das
Neves Rodrigues**

**Aprendizagem Automática Aplicada à Condução
de um Veículo com Direção Ackermann**

**Deep Learning Applied to Ackermann Steering
Vehicle Driving**



**João Gabriel das
Neves Rodrigues**

**Aprendizagem Automática Aplicada à Condução
de um Veículo com Direção Ackermann**

**Deep Learning Applied to Ackermann Steering
Vehicle Driving**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Artur José Carneiro Pereira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor José Nuno Panelas Nunes Lau, Professor Auxiliar no Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Luís Filipe de Seabra Lopes

Professor Associado da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Luis Paulo Gonçalves dos Reis

Professor Associado da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor Artur José Carneiro Pereira

Professor Auxiliar da Universidade de Aveiro

**agradecimentos /
acknowledgements**

Agradeço aos meus orientadores, Professor Artur Pereira e Professor Nuno Lau, e meus colaboradores, David Simões e João Bernardo Alves por me terem ajudado, corrigido e transmitido conhecimento para a criação desta dissertação. Agradeço também aos restantes elementos do IRIS-Lab por me terem apoiado durante a criação desta dissertação.

Palavras Chave

Aprendizagem Automática, Aprendizagem por Reforço, Aprendizagem Supervisionada, Condução Autônoma

Resumo

Nos últimos anos, tem havido um grande crescimento na área da Inteligência Artificial, principalmente por causa da aprendizagem automática. Esse crescimento é devido ao maior poder de processamento dos computadores atuais, o que permite a implementação de algoritmos já projetados, e à pesquisa e desenvolvimento de novos algoritmos. Esses algoritmos permitem a execução de tarefas difíceis de descrever diretamente pelos programadores. A aprendizagem é baseada em conjuntos de dados criados por seres humanos ou pelos próprios algoritmos, usando uma abordagem de tentativa e erro. A condução autônoma é uma das áreas em que os algoritmos de aprendizagem automática têm sido utilizados, com muitos dos avanços nessa área sendo resultado do seu uso.

O ROTA é um veículo robótico de pequenas dimensões usado para investigação e ensino de algoritmos de condução autônoma, que participa no Festival Nacional de Robótica. O cenário de atuação do veículo corresponde a uma pista com semelhanças marcantes a uma estrada convencional, contendo obstáculos, sinais de trânsito e zonas de estacionamento. Até ao início desta dissertação, as várias tarefas que o veículo tinha que autonomamente realizar eram codificadas diretamente pelo programador. O objetivo desta dissertação foi o de aplicar aprendizagem automática na implementação do módulo de condução, responsável por manter o veículo dentro da faixa de rodagem. Pretendia-se a aplicação de técnicas de aprendizagem supervisionada e de aprendizagem por reforço.

Para a aprendizagem supervisionada, optou-se por uma rede neuronal convolucional, tendo como entrada uma região de interesse da imagem da estrada captada por uma câmara montada na parte frontal do veículo e como saída a velocidade angular a aplicar para controlo do movimento. O conjunto de dados de treino foi obtido controlando manualmente o movimento do veículo com a ajuda de um comando e armazenando as imagens captadas pela câmara e as velocidades angulares dadas pelo humano. A recolha foi realizada num ambiente de simulação do veículo e da pista já existente. A rede treinada mostrou um bom desempenho quer no ambiente de simulação, quer com o veículo real na pista real durante o Robotica 2018. Para a aprendizagem por reforço, escolheram-se dois algoritmos: o DDQN e o DDPG. Para treinar estes algoritmos foi necessário proceder a alterações no ambiente de simulação de modo a permitir a interação. Os vários algoritmos desenvolvidos foram avaliados, fazendo-se uma análise comparativa. Os resultados alcançados com o algoritmo DDQN foram melhores que os alcançados com o DDPG, tendo sido possível conduzir no ambiente de simulação, embora com algumas oscilações e saídas parciais de pista.

Keywords

Machine Learning, Reinforcement Learning, Supervised Learning, Autonomous Driving

Abstract

In the last years, there has been a great increase in the area of Artificial Intelligence, mostly because of Machine Learning. This growth is due to higher processing power of current computers, which allows the implementation of already designed algorithms, and to the research and development of new ones. These algorithms allow the execution of tasks difficult to describe directly by programmers. Learning is based on data sets created by humans, or by the algorithms themselves, using a trial and error approach. Autonomous driving is one of the areas where machine learning algorithms have been used, with many of the advances in this area being a result of its use.

The ROTA is a small robotic vehicle used in research and teaching of autonomous driving algorithms, that participates in the Portuguese Robotics Open. The competing scenario corresponds to a track with strong similarities to a conventional road, containing obstacles, traffic signs and parking areas. Until the beginning of this dissertation, the various tasks that the vehicle had to carry out were coded directly by the programmer. The objective of this dissertation was to apply deep learning techniques in the implementation of the driving module, responsible for keeping the vehicle within the lane. The aim was to apply both supervised learning and reinforcement learning techniques. For supervised learning, a convolutional neural network was chosen, having as input a region of interest of the road image captured by a camera mounted on the front of the vehicle and as output the angular velocity to be applied to control the movement. The training data set was obtained by manually controlling the movement of the vehicle with the help of a command and storing the images captured by the camera and the angular velocities given by the human. The data gathering was performed in already existing simulation environment of the vehicle and track. The trained network showed a good performance both in the simulation environment and with the real vehicle on the real track during Robotica 2018. For reinforcement learning, two algorithms were chosen: DDQN and DDPG. To train these algorithms it was necessary to make changes in the simulation environment in order to allow interaction. The various algorithms developed were evaluated and a comparative analysis is presented. The results achieved using the DDQN algorithm were better than those achieved with DDPG, and it was possible to drive in the simulation environment despite of some oscillations and partially exiting the lane a few times.

Conteúdo

Conteúdo	i
Lista de Figuras	v
Lista de Tabelas	vii
Siglas	ix
1 Introdução	1
1.1 Motivação	1
1.2 Objetivos	2
1.3 Estrutura	2
2 Projeto ROTA	5
2.1 Prova de Condução Autônoma do Festival Nacional de Robótica	5
2.2 O Veículo	7
2.2.1 Arquitetura do Sistema de Controlo	8
2.3 Ambiente de Simulação	11
2.4 ROS	12
2.4.1 Comunicação	12
2.4.2 Ferramentas de Visualização e Depuração do Sistema	14
2.5 Gazebo	14
2.5.1 Física	15
2.5.2 Gráficos	15
2.5.3 Sensores	16
2.5.4 <i>Plugins</i>	16

2.5.5	Interface Gráfica	16
2.5.6	Objetos	17
3	Aprendizagem Automática	19
3.1	Aprendizagem Supervisionada	20
3.1.1	Redes Neurais Artificiais	20
3.1.2	Redes Neurais Profundas	27
3.2	Aprendizagem por Reforço	30
3.2.1	<i>Markov Decision Process</i>	31
3.2.2	<i>Q Learning</i>	32
3.2.3	<i>Deep Q Network</i>	32
3.2.4	<i>Double Deep Q Learning</i>	34
3.2.5	<i>Deep Deterministic Policy Gradient</i>	35
3.3	Condução Autônoma e Aprendizagem Automática	36
3.4	Tecnologias	37
3.4.1	TensorFlow	37
3.4.2	Keras	39
3.4.3	TFLearn	39
4	Condutor Autônomo	41
4.1	Piloto implementado com Aprendizagem Supervisionada	41
4.1.1	Construção do <i>Dataset</i>	42
4.1.2	Pré-processamento da imagem	44
4.1.3	Arquitetura da Rede Neuronal	45
4.1.4	Treino	46
4.1.5	Utilização do Modelo	46
4.1.6	Resultados	46
4.2	Piloto implementado com Aprendizagem por Reforço	48
4.2.1	Controlador do Ambiente de Simulação	49
4.2.2	Representação dos Espaços dos Estados e das Ações	51
4.2.3	Arquitetura dos Modelos	51
4.2.4	Hiperparâmetros	54
4.2.5	Aprendizagem	55
4.2.6	Resultados	56
5	Conclusão	63
5.1	Trabalho Futuro	64
A	Certificado da Classificação na Prova de Condução Autônoma	65

Lista de Figuras

2.1	Pista da competição	6
2.2	Sinais de Tráfego Verticais	7
2.3	Veículo ROTA	8
2.4	Arquitetura do Sistema de Controlo do ROTA	9
2.5	Máquina de Estados do Copiloto	10
2.6	Elementos do Ambiente de Simulação	11
2.7	Pistas do Ambiente de Simulação	12
2.8	Ambiente de Simulação	12
2.9	Interação com tópicos	13
2.10	Interação com serviços	14
2.11	Interface Gráfica do Gazebo	17
3.1	Nó de uma ANN	21
3.2	Gráfico da função sigmóide	21
3.3	Arquitetura de uma ANN	22
3.4	Momentum	25
3.5	Gráfico da função ReLU	27
3.6	Gráfico da função ELU	28
3.7	Exemplo de <i>Overfitting</i>	28
3.8	Aplicação de <i>dropout</i>	29
3.9	Operação de Convolução	30
3.10	Operação de Redução	30
3.11	<i>Reinforcement Learning</i> : Visão Geral	31
3.12	Exemplo de um grafo de Tensorflow	38
3.13	Estruturas dos sistemas de TensorFlow	38

4.1	Pista usada na criação do <i>dataset</i>	42
4.2	Exemplos do <i>dataset</i>	43
4.3	Histograma da velocidade angular dos exemplos do <i>dataset</i>	44
4.4	Arquitetura da CNN do piloto	45
4.5	Gráficos da Função Custo do Treino do Modelo SL	47
4.6	Pista de Teste	47
4.7	Imagem de entrada da Rede Neuronal Artificial (ANN) no mundo real	48
4.8	Arquitetura do Controlador de Ambiente	49
4.9	Pistas usadas na simulação	50
4.10	Arquitetura da ANN do piloto implementado com DDQN	52
4.11	Arquitetura da ANN da função ator do piloto implementado com DDPG	52
4.12	Arquitetura da ANN da função ator do piloto implementado com DDPG	53
4.13	DDQN: Evolução da Recompensa nos Episódios de Avaliação	57
4.14	DDQN: Evolução da Função de Custo nos Episódios de Treino	57
4.15	DDQN utilizando a segunda função de custo: Evolução da Recompensa nos Episódios de Avaliação	58
4.16	DDQN utilizando a segunda função de custo: Evolução da Função de Custo nos Episódios de Treino	59
4.17	DDPG: Evolução da Recompensa nos Episódios de Avaliação	60
4.18	DDPG: Evolução da Função de Custo nos Episódios de Treino	60

Lista de Tabelas

2.1	Sinais dos Painéis de Sinalização	6
4.1	Arquitetura da CNN do piloto	46
4.2	Arquitetura da rede neuronal do piloto implementado com DDQN	52
4.3	Arquitetura da rede neuronal da função ator	53
4.4	Arquitetura da rede neuronal da função crítico	54
4.5	Hiperparâmetros de treino de DDQN	54
4.6	Hiperparâmetros de treino de DDQN	55

Siglas

- Adam** *Adaptive Moment Estimation* 23, 26, 46
- ALVINN** *Autonomous Land Vehicle In a Neural Network* 36
- ANN** Rede Neuronal Artificial v, vi, 20–25, 27, 29, 32–37, 39, 41, 42, 44–46, 48, 49, 51, 59, 63
- API** *Application Programming Interface* 13, 38
- BGD** *Batch Gradient Descent* 23, 24
- CAN** *Controller Area Network* 8
- CNN** Rede Neuronal Convolutacional 29, 36, 41, 42, 45–47, 52, 63
- CPU** *Central Processing Unit* 37, 38
- CV** Computação Visual 29
- DART** *Dynamic Animation and Robotics Toolkit* 15
- DDPG** *Deep Deterministic Policy Gradient* 20, 30, 35, 37, 48, 49, 51, 54, 55, 59, 61, 64
- DDQN** *Double Deep Q Learning* 20, 30, 34, 35, 48, 49, 51, 53–55, 57, 58, 61, 63, 64
- DNN** Rede Neuronal Profunda 20, 27–29
- DPG** *Deterministic Policy Gradient* 35
- DQN** *Deep Q Network* 20, 30, 32–35, 37, 48, 49
- ELU** *Exponential Linear Unit* v, 27, 28
- FNR** Festival Nacional de Robótica 1, 2, 5, 11, 41, 47, 59, 63, 64
- GAN** *Generative Adversarial Network* 43

GD *Gradient Descent* 23, 24

GPS *Global Positioning System* 16

GPU *Graphics Processing Unit* 2, 37, 38

HAL *Hardware Abstraction Layer* 8–10

ICARSC *International Conference on Autonomous Robot Systems and Competitions* 5

IEETA *Instituto de Engenharia Electrónica e Telemática de Aveiro* 1

IMU *Inertial Measurement Unit* 16

IRIS-Lab *Intelligent Robotics and Intelligent Systems Laboratory* 1, 5

KNN *K Nearest Neighbour* 20

LLI *Low-Level Infrastructure* 8

LRF *Laser Range Finder* 8, 10, 16, 64

MBGD *Mini-batch Gradient Descent* 23, 24

MDP *Markov Decision Process* 20, 30, 31

ML *Machine Learning* 19, 20, 30, 36, 37, 39, 41

MSE *Mean Squared Error* 22, 46

NFQ *Neural Fitted Q Iteration* 33

ODE *Open Dynamics Engine* 15

OGRE *Object-oriented Graphics Rendering Engine* 15

ReLU *Rectified Linear Unit* v, 27

RL *Reinforcement Learning* 11, 19, 20, 30, 31, 35, 37, 48, 49, 51, 63, 64

RMS *Root Mean Squared* 26

ROS *Robot Operating System* 2, 8, 12–14, 41, 46, 49

SGD *Stochastic Gradient Descent* 23–25

SL *Supervised Learning* 19, 20, 30, 36, 41, 48, 51, 52, 63, 64

SPR *Sociedade Portuguesa de Robótica* 1, 5

SVM *Support Vector Machine* 20

UL *Unsupervised Learning* 19

USB *Universal Serial Bus* 8

VC Visão Computacional 29, 36

Introdução

1.1 MOTIVAÇÃO

Esta dissertação foi desenvolvida no âmbito do projeto ROTA, um projeto do *Intelligent Robotics and Intelligent Systems Laboratory* (IRIS-Lab), do Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA), que se dedica a tópicos de investigação relacionados com condução autónoma. No contexto deste projeto, foi projetado e desenvolvido um veículo autónomo de pequenas dimensões, que participa na prova de Condução Autónoma do Festival Nacional de Robótica (FNR), um evento anual com competições robóticas organizado pela Sociedade Portuguesa de Robótica (SPR). Nesta competição, cujo cenário de competição é uma pista com semelhanças marcantes com uma estrada convencional mas de menores dimensões, os participantes têm de construir um veículo autónomo capaz de ultrapassar diferentes desafios, tais como condução em estrada, estacionamento, identificação e observação de sinalização, e evitação de obstáculos.

O desenvolvimento destes veículos autónomos de pequenas dimensões facilita a criação de algoritmos capazes de conduzir no mundo real. Os veículos autónomos têm surgido nos últimos anos como os veículos da marca *Tesla*¹, os protótipos da *Waymo*² e *Nvidia* [1] e de algumas universidades, como o AtlasCar³ [2], da Universidade de Aveiro. Alguns destes veículos são dotados de algoritmos de aprendizagem automática, capazes de aprender sozinhos a conduzir no mundo real de forma segura, sem causar danos quer materiais, quer pessoais.

No âmbito do mesmo projeto, e dada a impossibilidade de se construir uma pista real no laboratório, foi desenvolvido um ambiente de simulação realista, incluindo o veículo ROTA, a pista e os diversos elementos que nela existem ou podem ser colocados. Estes ambientes de simulação são essenciais, porque permitem o desenvolvimento e teste de soluções sem pôr em causa a integridade do veículo e dos elementos do ambiente envolvente.

¹<https://www.tesla.com/autopilot>. Acedido em 28/05/2018

²<https://waymo.com/>. Acedido em 28/05/2018

³<http://atlas.web.ua.pt/>. Acedido em 28/05/2018

1.2 OBJETIVOS

Antes do desenvolvimento desta dissertação, o veículo ROTA já era dotado de um sistema de controlo de condução capaz de realizar a navegação num circuito fechado. Este sistema tem como base algoritmos de visão por computador tradicionais e localização global e relativa à faixa de rodagem. No entanto, estes algoritmos são difíceis de implementar e de melhorar. Nestes últimos anos, tem havido um grande aumento da popularidade, e consequentemente da pesquisa e desenvolvimento de técnicas de aprendizagem automática, capazes de aprender e realizar tarefas cujos algoritmos são difíceis de desenvolver, como por exemplo, classificação de imagens [3]–[5], processamento e classificação de texto [6], [7] e processamento de voz [8]. Este crescimento de popularidade deve-se ao aumento da capacidade de processamento do *hardware*, nomeadamente de *Graphics Processing Units* (GPUs), e à disponibilização de bibliotecas, que facilitam o desenvolvimento destes algoritmos e aceleram a sua execução, e ao aparecimento de novos algoritmos.

O objetivo principal desta dissertação era a aplicação de técnicas de aprendizagem automática no desenvolvimento de um módulo de condução do veículo ROTA, que permitisse ao veículo manter-se a circular dentro da faixa de rodagem. Para o desenvolvimento deste módulo, devem ser exploradas técnicas de aprendizagem supervisionada e de aprendizagem por reforço.

1.3 ESTRUTURA

Este documento encontra-se dividido em cinco capítulos. O capítulo 1, onde esta secção se encontra, refere-se à introdução deste documento, onde se descreve a motivação e os objetivos deste trabalho.

O capítulo 2 descreve o que é o projeto ROTA, a Prova de Condução Autónoma do FNR à qual esse projeto participa, o ambiente de simulação utilizado para o desenvolvimento deste trabalho, e duas ferramentas utilizadas para o desenvolvimento do projeto, o *Robot Operating System* (ROS), como plataforma de desenvolvimento de controladores para robótica, e o *Gazebo*, utilizado para desenvolver o ambiente de simulação.

O capítulo 3 descreve o que é aprendizagem automática e os seus ramos, focando-se mais nos dois ramos explorados neste trabalho, aprendizagem supervisionada e aprendizagem por reforço. A descrição destes dois ramos, são apresentadas várias técnicas utilizadas para o desenvolvimento deste trabalho. Também apresenta várias aplicações de aprendizagem automática na condução autónoma, quer utilizando aprendizagem supervisionada, quer utilizando aprendizagem por reforço. Finalmente, este capítulo apresenta várias ferramentas normalmente utilizadas para o desenvolvimento de algoritmos de aprendizagem automática.

O capítulo 4 descreve o trabalho realizado para o desenvolvimento do módulo de condução do veículo ROTA, utilizando técnicas de aprendizagem supervisionada e aprendizagem por reforço. Para o módulo de condução implementado com aprendizagem supervisionada, é apresentado como foi construído o conjunto de dados, as técnicas utilizadas para aumentar este conjunto de dados, a arquitetura do modelo utilizado, o treino do algoritmo e a utilização deste

modelo, quer no ambiente de simulação, quer no ambiente real, e, finalmente, são mostrados os resultados obtidos. Para os módulos de condução implementados com aprendizagem por reforço, é descrito o controlador do ambiente de simulação utilizado para treinar o algoritmos, como foram representados os espaços dos estados e das ações e a justificação dessa escolha, as arquiteturas dos modelos utilizados pelos módulos de condução e seus hiperparâmetros, como se treinou os modelos, e, por último, os resultados obtidos e a comparação entre os módulos de condução implementados com aprendizagem por reforço.

Finalmente, o capítulo 5 contém as conclusões obtidas do desenvolvimento do trabalho desenvolvido nesta dissertação. Nestas conclusões, está a comparação entre os vários módulos de condução descritos no capítulo 4. Também faz parte deste capítulo a descrição de trabalho que poderá ser realizado no futuro, para melhorar e complementar o trabalho já realizado.

Projeto ROTA

O projeto ROTA é um projeto do IRIS-Lab que promove a investigação na área da condução autónoma. O objetivo deste projeto é ensinar a investigação e desenvolvimento de um veículo autónomo, de pequenas dimensões, capaz de resolver alguns desafios como navegar num circuito fechado, com semelhanças marcantes a uma estrada convencional. O trabalho desenvolvido é demonstrado em provas de condução autónoma no FNR.

Neste capítulo é descrita a prova de condução em que este projeto participa (secção 2.1), o veículo e a arquitetura do seu sistema de controlo (secção 2.2), o ambiente de simulação que potencia o teste de soluções que depois podem ser portadas para o ambiente real (secção 2.3) e tecnologias usadas para o desenvolvimento deste projeto (secções 2.4 e 2.5).

2.1 PROVA DE CONDUÇÃO AUTÓNOMA DO FESTIVAL NACIONAL DE ROBÓTICA

O FNR é um encontro que “promove a Ciência e Tecnologia junto dos jovens, professores, investigadores e do público em geral, através de competições de robôs autónomos”¹, organizado anualmente em Portugal pela SPR [9]. Para além destas competições, inclui um Encontro Científico (*International Conference on Autonomous Robot Systems and Competitions* (ICARSC)) onde são apresentados resultados de investigações na área da robótica.

Entre as várias competições presentes neste festival, encontra-se a competição de Condução Autónoma, em que o projeto ROTA participa. Nesta competição, os veículos autónomos participantes têm de navegar num circuito fechado, com desafios que simulam situações encontradas na condução de um automóvel convencional, como não sair da faixa de rodagem, evitar a colisão com obstáculos, estacionar e identificar sinais.

O circuito, que em 2018 tinha o formato de B (figura 2.1), simula uma estrada com uma faixa de rodagem composta por duas vias, uma passadeira e painéis de sinalização (tabela 2.1) no início do percurso dividindo o circuito em dois setores e áreas de estacionamento, uma com dois lugares para estacionamento perpendicular e uma outra para estacionamento paralelo. Nos vários desafios, podem ser ainda adicionados um túnel, uma zona de obras delimitada por

¹<https://robotica2018.festivalrobotica.pt>. Acedido em 28/05/2018

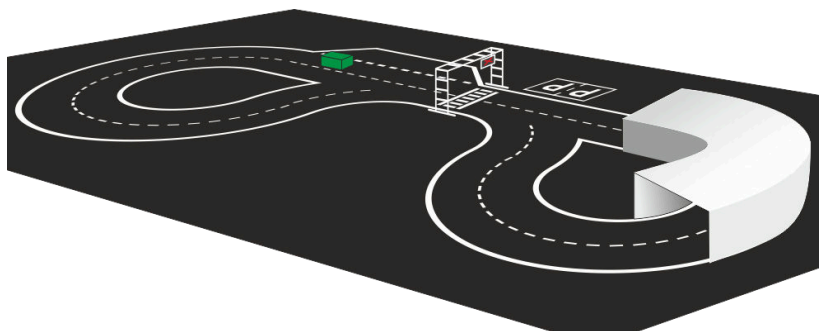


Figura 2.1: Pista da competição. Fonte: ²

Tabela 2.1: Sinais dos Painéis de Sinalização. Adaptado de: ²

Função	Ação	Sinal
1	Seguir à esquerda	←
2	Seguir à direita	→
3	Seguir em frente	↑
4	Parar	×
5	Fim de prova	Checkerboard pattern
6	Estacionar	P

cones de sinalização, outros obstáculos e sinais de trânsito verticais (figura 2.2). Os painéis de sinalização (tabela 2.1) indicam qual a ordem que o veículo deve obedecer.

A prova de condução autónoma é constituída por desafios pertencentes a três categorias: desafios de condução, desafios de estacionamento e desafios de deteção de sinalização de tráfego vertical.

Existem quatro desafios de condução, D1 a D4, com um grau de dificuldade incremental. Na prova D1, o veículo parte imediatamente antes da passadeira ao sinal de partida (seta verde que aponta para cima), percorre o circuito duas vezes e para imediatamente antes da

²https://robotica2018.festivalrobotica.pt/files/documents/fnr2018_autonomous_driving-2.pdf. Acedido em 09/04/2018



Figura 2.2: Sinais de Tráfego Verticais. Fonte: ³

passadeira. Na prova D2, o veículo deve percorrer o circuito duas vezes obedecendo às ordens dos painéis de sinalização, ou seja, seguindo a direção indicada pelo painel ou parando por um período. Na prova D3, são adicionados o túnel no primeiro setor e dois obstáculos no segundo setor do circuito, com os quais o veículo deve evitar colidir. Na prova D4, para além dos obstáculos e do túnel, é adicionada uma zona de obras no primeiro setor.

As provas de estacionamento dividem-se em dois tipos, estacionamento perpendicular, provas B1 (sem obstáculo) e B2 (com obstáculo), e paralelo, provas P1 (sem obstáculos) e P2 (com obstáculos). Nesses desafios, o veículo deve estacionar numa das áreas de estacionamento próprias. Nas provas de estacionamento perpendicular, é necessário efetuar um dos desafios de condução, ao contrário dos desafios de estacionamento paralelo. Na prova de estacionamento perpendicular com obstáculo (B2), este é colocado numa das áreas de estacionamento, obrigando o veículo a estacionar na outra área. No desafio de estacionamento paralelo com obstáculos (P2), estes são dispostos na área de estacionamento sem uma posição específica. No entanto, a distância entre estes não deve ser inferior a duas vezes o comprimento do veículo em prova. O veículo deve parar paralelamente à faixa de rodagem.

Na prova de deteção de sinais verticais (V1), são dispostos próximo da faixa de rodagem seis sinais verticais dos doze mostrados na figura 2.2, dois por tipo. O objetivo é identificar o tipo de sinal e o sinal em si.

2.2 O VEÍCULO

O ROTA (figura 2.3), é um veículo retangular com 60 cm de comprimento por 45 cm de largura. É um triciclo, com uma roda atrás sem direção e com tração e duas rodas à frente com direção e sem tração, implementando o modelo de locomoção Ackermann. O sistema de

³https://robotica2018.festivalrobotica.pt/files/documents/fnr2018_autonomous_driving-2.pdf. Acedido em 09/04/2018

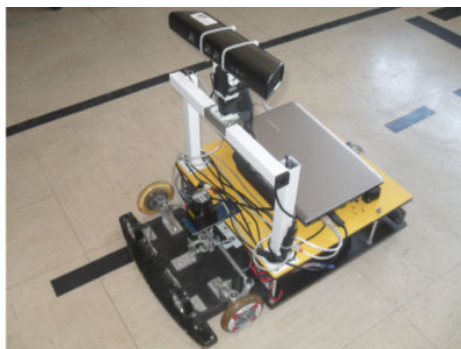


Figura 2.3: Veículo ROTA. Fonte: [10]

tração é baseado num motor dotado de um *encoder* e a direção é baseado num servo motor com *feedback*.

Dois dos principais sensores de percepção são uma câmara RGB-D e um *Laser Range Finder* (LRF). A câmara RGB-D permite capturar imagens para a navegação do veículo e identificação de sinais. Esta câmara está montada num suporte com dois graus de liberdade, horizontal (*pan*) entre $[-180, 180]$ graus, e vertical (*tilt*) entre $[-90, 90]$ graus. O LRF, situado à frente do veículo, permite detetar obstáculos, de maneira a evitar colisões.

O sistema de controlo do veículo está estruturado em duas camadas. Há uma camada de baixo nível, baseada numa rede *Controller Area Network* (CAN) de microcontroladores, situados na proximidade de atuadores e sensores. O controlo de alto nível corre num computador portátil e comunica com a cama de baixo nível através de uma ligação série, usando uma porta *Universal Serial Bus* (USB).

O software de alto nível é desenvolvido em cima do ambiente ROS [11], um *middleware* muito usado em robótica que permite que os vários componentes comuniquem entre si. Este ambiente também disponibiliza um grande número de bibliotecas de desenvolvimento e ferramentas para visualização, teste e depuração do sistema.

2.2.1 Arquitetura do Sistema de Controlo

A arquitetura do sistema que controla este veículo é apresentada na figura 2.4. Os blocos retangulares delimitados por linha simples representam dados (conjuntos de tópicos ROS), os blocos retangulares delimitados por linha dupla representam dispositivos e os blocos ovais representam módulos de processamento.

Os três blocos de dispositivos representam o LRF, a câmara RGB-D (Kinect) e a camada de baixo nível (ou *Low-Level Infrastructure* (LLI)) que representa o conjunto de sensores e atuadores.

O módulo da camada de Abstração de *Hardware* ou (*Hardware Abstraction Layer* (HAL)) é responsável pela comunicação com a camada de baixo nível e por criar uma abstração à interação entre módulos de alto nível e de baixo nível, permitindo, por exemplo, a obtenção de dados para a odometria e posição dos servos.

O módulo *ImageAcq* é responsável pela aquisição das imagens da câmara e pela sua publicação ou das suas derivadas. Estas imagens são processadas por dois módulos. O

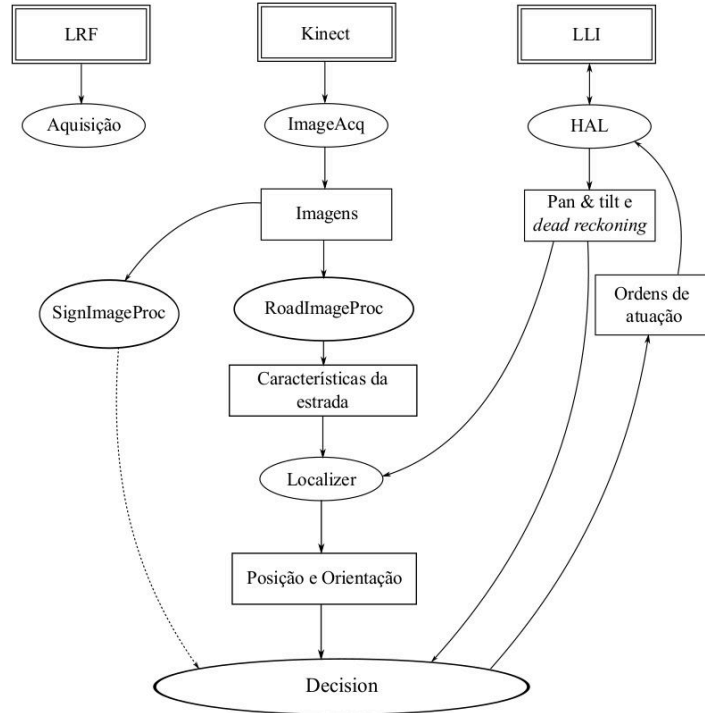


Figura 2.4: Arquitetura do Sistema de Controlo do ROTA. Fonte: [10]

módulo **RoadImageProc** extrai informação para a localização do veículo na faixa de rodagem, como por exemplo, as linhas delimitadoras das vias de circulação e a passadeira. Esta informação é transmitida ao módulo de localização. O módulo **SignImageProc** é responsável pela identificação dos sinais representados nos painéis sinaléticos e verticais.

O módulo **Localizer** é responsável por determinar uma estimativa da localização e orientação, utilizando a estimativa anterior, os dados provenientes do módulo **HAL** e a informação proveniente do módulo **RoadImageProc**.

O módulo **Decision** representa a entidade responsável pelo controlo do veículo de maneira a resolver os desafios que lhe são colocados. Este módulo divide-se em outros dois, o Copiloto e o Piloto.

O Copiloto é a entidade responsável pela navegação e pelos comportamentos de condução que o Piloto deve tomar. Esta entidade está implementada sob a forma de uma máquina de estados. Um exemplo é a máquina de estados utilizada para superar os desafios D1 a D3, representada pela figura 2.5.

O estado **Init** é o estado inicial do robô. Nele espera-se que todos os serviços e tópicos fiquem disponíveis antes de avançar para o estado seguinte (**Idle**).

O estado **Idle** é um estado de espera. Neste, espera-se pela inicialização dos nós de localização e pela ordem de começo de prova, dando-se a transição para o estado **At Crosswalk**.

No estado **At Crosswalk**, o veículo encontra-se parado imediatamente antes da passadeira e é identificado o sinal mostrado no painel sinalético. Para transitar para o próximo estado, é necessário identificar um sinal diferente do sinal stop. Depois da identificação desse sinal, a câmara é posicionada de maneira a ser possível identificar as linhas na pista.

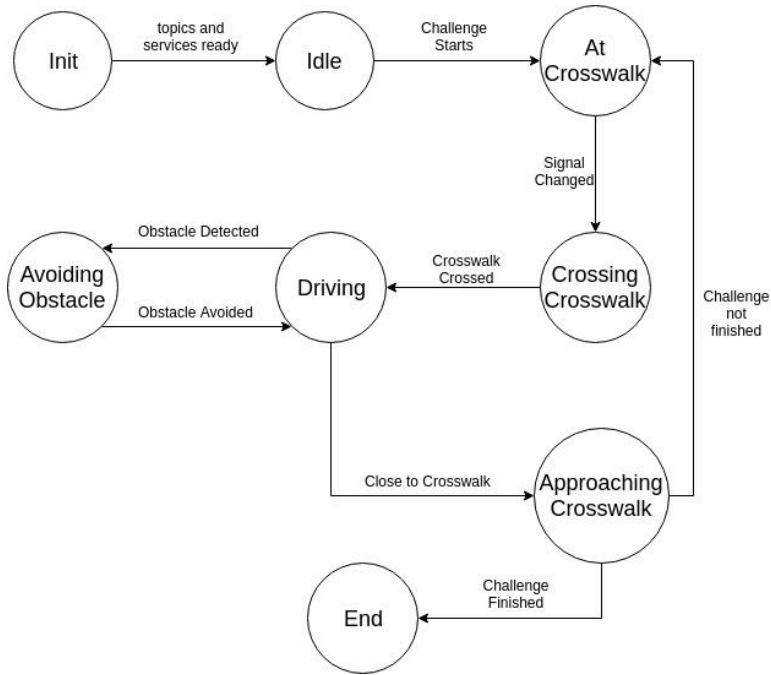


Figura 2.5: Máquina de Estados do Copiloto. Adaptado de: [10]

No estado **Crossing Crosswalk**, o veículo atravessa a passadeira, corrigindo a orientação de modo a ficar orientado com a via, antes de transitar para o próximo estado.

No estado **Driving**, o robô conduz numa via e orientado segundo esta. Pode transitar para o estado **Avoiding Obstacle**, no caso de detetar um obstáculo na via em que circula, ou para o estado **Approaching Crosswalk** caso se encontre próximo da passadeira.

No estado **Avoiding Obstacle**, o veículo muda de via de maneira a evitar o obstáculo detetado e circula nesta uma distância predeterminada, de forma a ultrapassar este obstáculo. Depois de percorrer esta distância, o veículo muda para a via onde estava a circular inicialmente e transita novamente para o estado **Driving**.

No estado **Approaching Crosswalk**, a câmara é posicionada de maneira a que se veja o que está imediatamente à frente da base do veículo. Assim, muda para um comportamento de condução que permita parar na passadeira, se for necessário detetar o sinal no painel. Após o avistamento desta, o veículo avalia se já completou a prova. Caso afirmativo, transita para o estado **End**, senão transita para o estado **At Crosswalk**.

A informação usada pelo Copiloto é proveniente dos módulos **Localizer** e **HAL**. A passadeira é detetada tendo em conta a localização do robô na pista e a sua forma. A deteção de obstáculos é feita a partir dos dados provenientes do **LRF**. Existe um obstáculo no caminho do robô se o **LRF** identificar um objeto numa posição que provocasse uma colisão caso o veículo siga a via em que circula no momento.

O Piloto é a identidade responsável pela condução do veículo, seguindo as ordens do Copiloto. Nos estados **Init** e **Idle**, o Piloto está desativado, visto que não é necessário conduzir nestes estados. Nos estados **Driving** e **Avoid Obstacle**, o Piloto utiliza um algoritmo de navegação que faz o seguimento da via. Esta navegação pode ser feita a partir de vários tipos

de informação, como a posição transversal relativa à faixa de rodagem, a posição global, a odometria ou diretamente da imagem recebida da câmara do veículo.

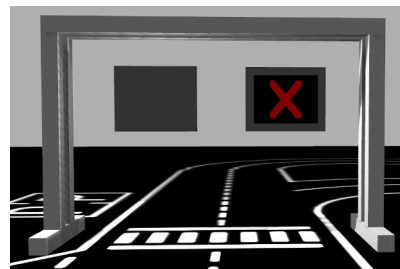
2.3 AMBIENTE DE SIMULAÇÃO

Para além do veículo físico, existe um ambiente de simulação. Este ambiente de simulação foi desenvolvido no contexto de uma dissertação de mestrado [10]. Este ambiente permite um desenvolvimento mais rápido, fácil e flexível, pois evita a criação do espaço fisicamente. Também é útil para o desenvolvimento de algoritmos de *Reinforcement Learning* (RL), pois permite reiniciar as condições iniciais, pausar e retomar a simulação (o que permite, por exemplo, cálculos mais complexos durante a simulação, impossíveis de realizar devido à perda de informação extraída pelos sensores durante este cálculo) e evita danos causados pelo veículo, por exemplo, durante o treino de um modelo de RL, onde, ao início, este modelo não sabe como controlar devidamente o veículo. Este ambiente de simulação foi criado com o simulador Gazebo (secção 2.5).

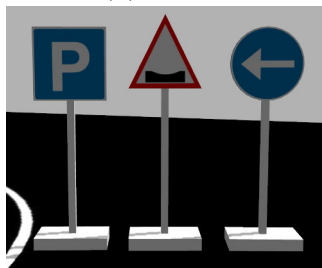
O ambiente simulado não só inclui o veículo (figura 2.6a), como também algumas pistas, incluindo as utilizadas na prova de condução autónoma do FNR (figura 2.7), o pórtico com os painéis de sinalização (figura 2.6b), sinais de trânsito (figura 2.6c) e obstáculos (figura 2.6d). Na figura 2.8 pode-se observar uma cena do ambiente simulado.



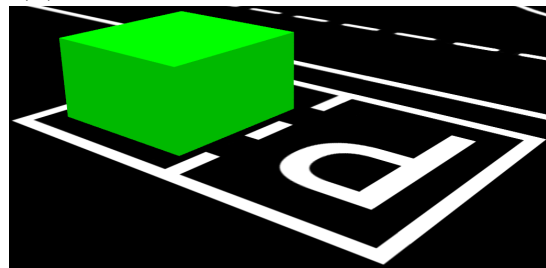
(a) ROTA



(b) Pórtico com painéis de sinalização



(c) Sinais Verticais



(d) Obstáculo

Figura 2.6: Elementos do Ambiente de Simulação

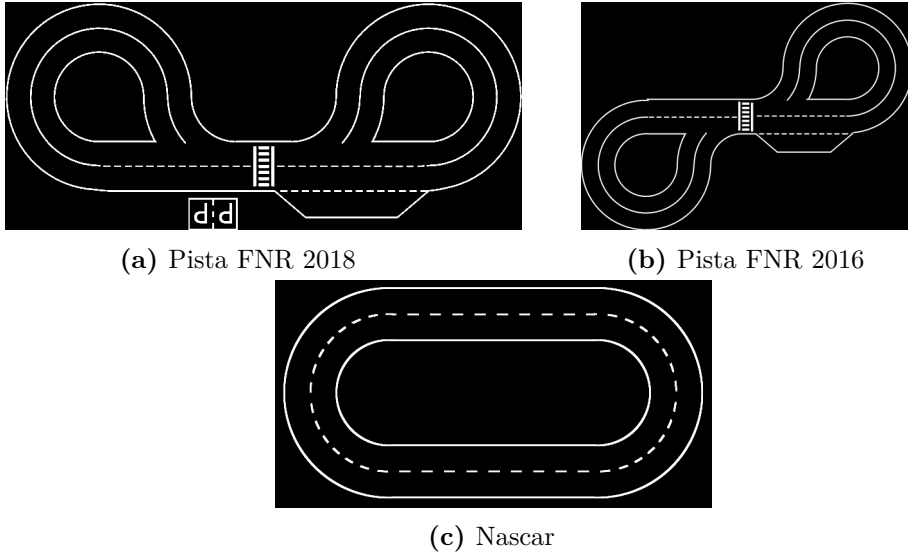


Figura 2.7: Pistas do Ambiente de Simulação

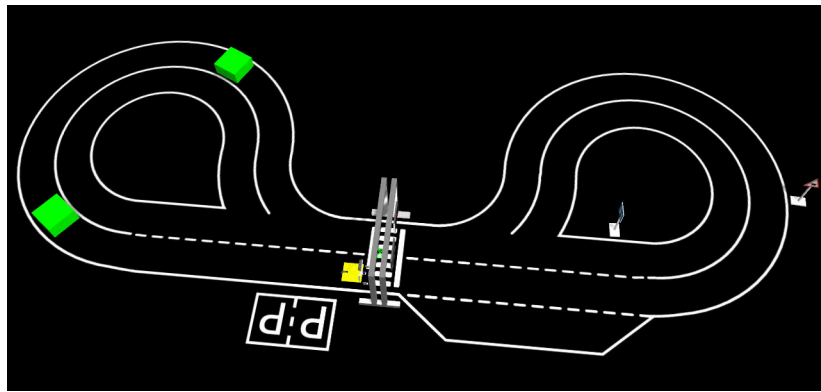


Figura 2.8: Ambiente de Simulação

2.4 ROS

ROS [11] é uma infraestrutura de código aberto usado para o desenvolvimento de *software* para a robótica. Esta infraestrutura fornece um meio de comunicação onde os vários processos, denominados nós, comunicam entre si. Também fornece um conjunto de bibliotecas, como controladores e algoritmos estado-de-arte, que apoiam a criação de novos conteúdos, evitando a criação de código a um nível mais baixo. Finalmente, fornece ferramentas de visualização, teste e depuração do sistema. As linguagens principais para a criação de código são o **Python** e **C++**.

2.4.1 Comunicação

A comunicação entre os vários nós pode ser feita de duas maneiras, através de tópicos, ou através de serviços.

A comunicação através de tópicos (figura 2.9) segue o paradigma de produtor/consumidor. Para que os nós comuniquem entre si através deste tipo de comunicação, o nó produtor regista-se no **ROS Master**, notificando que vai publicar mensagens de um certo tipo para um

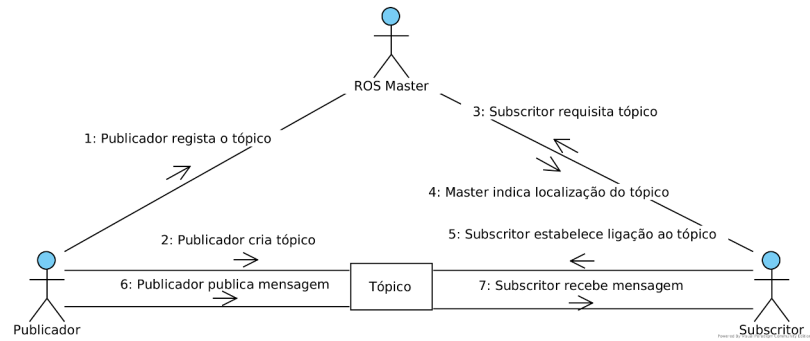


Figura 2.9: Interação com tópicos

determinado tópico. Os nós que requerem a informação dessas mensagens, os subscritores, registam-se no **ROS Master**, mostrando essa intenção. O **ROS Master** indica aos subscritores onde está o tópico que contém essa informação. Sabendo onde está o tópico, os subscritores contactam o(s) publicador(es), requerendo um canal de comunicação e negociam o protocolo de comunicação. A partir deste ponto, o(s) publicador(es) envia(m) a informação pretendida para os subscritores que subscreveram o tópico. Analisando estas interações, este tipo de comunicação é muito idêntico ao padrão de desenho do mediador, no entanto, a troca de mensagens em ROS é feita diretamente entre os nós, não através do mediador, cujo papel é informar onde estão os tópicos com a informação.

O ROS também permite a comunicação entre nós através de serviços (figura 2.10). Este tipo de comunicação segue o paradigma de cliente/servidor. À semelhança do paradigma anterior, o servidor regista-se no **ROS Master**, notificando a existência dos serviços que fornece. O cliente questiona o **ROS Master** sobre a existência dos serviços que necessita e, se existirem, este informa a sua localização. Quando necessário, o cliente envia um pedido ao servidor, que o atende e envia o resultado.

Estes dois tipos de comunicação tem algumas diferenças como o número de nós que podem comunicar entre si e a perda de pacotes. Em relação ao número de nós que podem comunicar entre si, os tópicos permitem uma comunicação um para muitos, enquanto que os serviços permitem uma comunicação um para um. Em relação à perda de pacotes, existe na comunicação através de tópicos, ao contrário da comunicação através de serviços.

Do ponto de vista do programador, o ROS fornece uma *Application Programming Interface* (API) que tornam os passos de comunicação transparentes ao programador, tornando mais fácil o desenvolvimento de algoritmos que necessitam de comunicação entre os diferentes processos, ou, neste caso, nós.

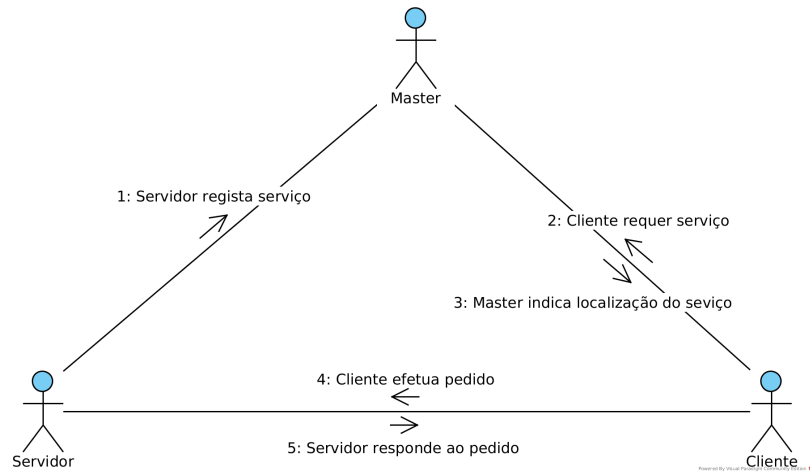


Figura 2.10: Interação com serviços

2.4.2 Ferramentas de Visualização e Depuração do Sistema

O ambiente ROS possui ferramentas de visualização e outras que ajudam no teste e depuração do sistema. Entre estas estão ferramentas de visualização do grafo do sistema (`rqt_graph`), visualização 3D (`rviz`), ferramentas de *logging* (`rqt_console`), gravação e reprodução (`rosbag` e `rqt_bag`) e um conjunto de ferramentas de linha de comandos para testar e depurar várias partes do sistema como, por exemplo, nós (`rostop`), tópicos e suas mensagens (`rostopic`), serviços (`rosservice`), parâmetros do sistema (`rosparam`), entre outras.

`rqt_graph` permite visualizar a arquitetura do sistema que está a ser executado: a ligação entre nós e tópicos e alguma informação sobre as mensagens.

`rviz` permite, por exemplo, visualizar num ambiente 3D o(s) robô(s), os dados provenientes dos seus sensores e sua(s) posição(ões).

`rqt_console` permite ver as mensagens de *log* que os vários nós criam, podendo ser filtradas por severidade da mensagem (i.e. *Debug*, *Info*, *Warn*, *Error* e *Fatal*), nó, tópico, tempo da publicação e/ou localização.

`rosbag` permite gravar e reproduzir mensagens de determinados tópicos. Também é possível tratar os dados recolhidos para outros fins, como a criação de *datasets*. `rqt_bag` permite não só a gravação e reprodução das mensagens, como também é possível visualizá-las.

As várias ferramentas de linha de comandos permitem ver os elementos do sistema, o seu tipo, mensagens transmitidas e informação destas.

2.5 GAZEBO

O **Gazebo** é um ambiente de simulação para aplicações robóticas, capaz de simular modelos de objetos (incluindo robôs) presentes no seu ambiente. Os objetos são tratados como um conjunto de partes rígidas interligadas por articulações. Cada parte inclui um modelo de colisões, uma representação visual e um modelo inercial. A cada parte podem ainda ser associado sensores.

O sistema **Gazebo** é dividido em duas partes, o servidor, **gzserver**, e o cliente, **gzclient**. O servidor é responsável por calcular a física, o *rendering* dos objetos e a informação captada pelos sensores. O cliente é responsável por fornecer uma interface gráfica ao utilizador que permite a visualização e interação com o ambiente de simulação.

2.5.1 Física

O motor de física é o responsável pela simulação das propriedades físicas dos objetos pertencentes ao ambiente de simulação. Estas vão desde as propriedades dos corpos até às leis físicas que modificam o estado e movimento desses corpos. Este motor de física utiliza quatro motores *open-source* já existentes, o *Open Dynamics Engine* (ODE)⁴, o *Bullet*⁵, o *Simbody*⁶ e o *Dynamic Animation and Robotics Toolkit* (DART)⁷. As características fornecidas por este motor são a deteção de colisões, tipos de movimento efetuado pelas articulações, massa e inércia.

As articulações estabelecem ligações entre as diferentes partes de um modelo, de forma a criar relações dinâmicas. Estas articulações podem ser do tipo prismático, de revolução do tipo 1 e 2, do tipo bola, engrenagens, pistão e universal.

As articulações prismáticas têm um grau de liberdade em termos de translação e permitem dois corpos deslocarem-se num dos eixos, limitados por um intervalo contínuo de valores.

As articulações de revolução 1 possuem um grau de liberdade em termos rotacionais e permitem a um corpo rodar sobre um eixo em relação ao seu centro de rotação. Por sua vez, as articulações de revolução 2 são duas articulações de revolução 1 em série, permitindo a rotação sobre dois eixos em simultâneo, podendo-se restringir ou não essa rotação.

Nas articulações do tipo bola, os corpos rodam apenas em relação ao seu centro de rotação, tendo três graus de liberdade em termos rotacionais.

As articulações do tipo engrenagem são compostas por articulações de revolução, que simulam rodas dentadas.

As articulações do tipo pistão são articulações com movimento de translação segundo um eixo, semelhante às articulações prismáticas, mas também possibilitam a execução de rotações segundo a direção do movimento de translação.

As articulações universais são semelhantes às articulações do tipo bola, exceto que restringem a um grau de liberdade.

2.5.2 Gráficos

O simulador **Gazebo** também utiliza um motor gráfico *open-source* já existente, o *Object-oriented Graphics Rendering Engine* (OGRE)⁸. Esta biblioteca é utilizada para realizar o *rendering* dos cenários 3D simulados. Estes cenários permitem apresentar ao utilizador uma interface gráfica com o estado do mundo. Também é utilizada pelas bibliotecas de sensores,

⁴<http://www.ode.org/>. Acedido em 28/05/2018

⁵<http://bulletphysics.org/wordpress/>. Acedido em 28/05/2018

⁶<https://simtk.org/projects/simbody/>. Acedido em 28/05/2018

⁷<https://dartsim.github.io/>. Acedido em 28/05/2018

⁸<https://www.ogre3d.org/>. Acedido em 28/05/2018

como câmaras, para criar as imagens captadas por estas. O *rendering* inclui a simulação da iluminação, das texturas e do céu.

2.5.3 Sensores

Os sensores são partes do modelo robô que criam informação a partir do ambiente virtual. O **Gazebo** já fornece uma biblioteca com várias categorias de sensores como altímetros, câmaras, sensores de contacto, profundidade, *Global Positioning System* (GPS), *Inertial Measurement Unit* (IMU), entre outros. É possível tornar mais realista a extração de informação adicionando ruído à informação recolhida. Para além dos sensores já fornecidos pelo **Gazebo**, é possível criar novos sensores utilizando *plugins*.

2.5.4 Plugins

É possível estender as funcionalidades do **Gazebo** através da criação de *plugins*. Estes *plugins* podem ser do tipo mundo, modelo, parte visual do modelo, sensor, sistema e interface gráfica.

Os *plugins* do mundo ligam-se a um determinado ambiente virtual, permitindo a alteração do seu estado, através da manipulação, adição, remoção e alteração de características de qualquer objeto presente neste ambiente.

Os *plugins* de modelo são responsáveis pela parte física dos objetos simulados, permitindo a aproximação da dinâmica destes ao comportamento do objeto real, pois é possível obter informações sobre o estado interno e atuar sobre as várias articulações.

Os *plugins* da parte visual são responsáveis pela parte visual dos objetos simulados, encontrando-se ligados a um modelo em específico, e permitem alterar o material dos corpos que constituem esse modelo.

Os *plugins* de sensores simulam a extração de informação proveniente do ambiente. Estes *plugins* podem simular câmaras, LRF, radar, contacto, entre outros. Também permitem criar novos sensores, para além dos já existentes.

Os *plugins* de sistema fornecem o controlo sobre o processo de inicialização do próprio **Gazebo**. Um exemplo é a gravação de imagens em disco produzidas dentro do ambiente de simulação.

Os *plugins* da interface gráfica permitem ao utilizador criar interfaces personalizáveis dentro do próprio **Gazebo**.

2.5.5 Interface Gráfica

A interface gráfica do cliente **Gazebo** (figura 2.11) é uma interface que permite ao utilizador não só a visualização do ambiente de simulação, como também a manipulação deste. Estas manipulações vão desde a adição e remoção de objetos, até à translação e rotação destes, alteração da iluminação e alteração da velocidade de simulação. Esta interface também permite a criação ou modificação dos modelos 3D, quer já existentes no formato utilizado pelo **Gazebo**, quer modelos criados por outras ferramentas.

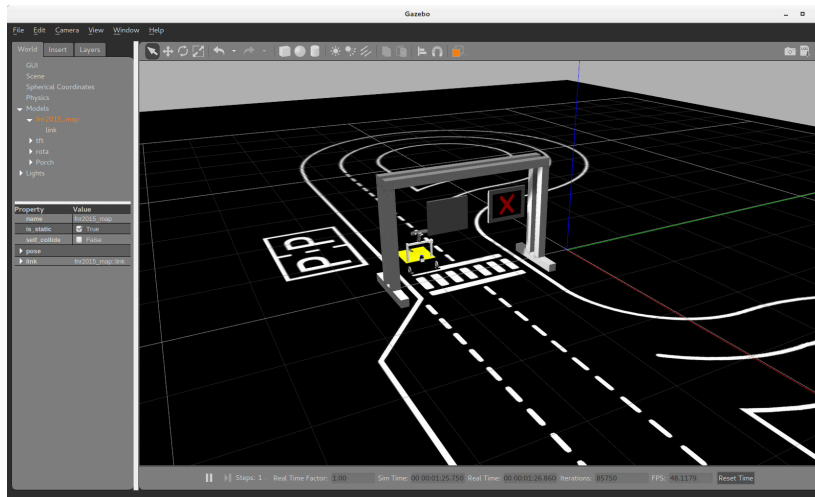


Figura 2.11: Interface Gráfica do Gazebo

2.5.6 Objetos

Os objetos presentes no ambiente de simulação são representados em formato **SDF**, um dialeto **XML**, que descreve a forma e texturas das suas partes, suas articulações, sensores e *plugins* do modelo. Na representação das partes do objeto estão incluídos atributos como a forma, texturas, propriedades de luminosidade e os modelos de colisão e inércia. Na representação das articulações estão presentes atributos como o tipo de articulação, as partes que esta interliga, os eixos de liberdade e restrições de movimento.

O mundo pode ser visto como um objeto que contém outros objetos, elementos de iluminação, *plugins* do mundo e propriedades globais.

As formas das partes dos objetos podem ser formas já existentes, ou especificadas em formatos externos como **STL**, **OBJ** ou **Collada**.

Aprendizagem Automática

Aprendizagem Automática (*Machine Learning* (ML)) é um ramo da Inteligência Artificial. Neste ramo, procura-se encontrar um modelo que resolva uma tarefa para a qual seja difícil construir um algoritmo. Alguns exemplos destas tarefas são classificações, onde o algoritmo indica qual ou quais categorias a que uma dada observação pertence, regressões, onde o algoritmo procura encontrar relações entre variáveis independentes (observação) e variáveis dependentes (resultado esperado) e até controlo de máquinas. Para obter os modelos, procura-se obter relações e/ou padrões a partir de um conjunto de dados (*dataset*) relacionado com a tarefa ou através de experiência ao realizar a tarefa.

Dentro de ML existem três ramos, dependendo do conjunto de dados fornecidos: aprendizagem supervisionada (*Supervised Learning* (SL)), aprendizagem não supervisionada (*Unsupervised Learning* (UL)) e aprendizagem por reforço (*Reinforcement Learning* (RL)).

Em SL, o modelo é obtido a partir de um *dataset* com exemplos já etiquetados, ou seja, que não só contém as amostras, como também o resultado esperado. Exemplo de um *dataset* é o MNIST¹, que contém um conjunto de imagens de dígitos manuscritos e a que dígito corresponde.

Em UL, os exemplos do *dataset* não contêm as etiquetas. Um dos objetivos é agregar os exemplos em conjuntos para se poder visualizar melhor os exemplos e classificá-los. Um exemplo é a agregação das imagens pertencentes ao *dataset* MNIST e a classificação os conjuntos obtidos.

Em RL, não existe um *dataset* construído *a priori*. No entanto, é a partir da experiência, ao tentar resolver o problema, que se obtém o modelo. As ações tomadas são avaliadas através de uma função de recompensa, que pontua esta ação avaliando vários aspetos, como por exemplo a ação tomada e o estado a que esta ação levou. Esta função recompensa ou penaliza o agente, consoante a ação tomada o aproxima ou não do objetivo. O agente aprende as ações a tomar maximizando as recompensas recebidas. Exemplos de problemas que se podem resolver com RL são o conjunto de problemas da Gym² da OpenAI, em que se controla um

¹<http://yann.lecun.com/exdb/mnist/>. Acedido em 28/05/2018

²<http://gym.openai.com/>. Acedido em 16/05/2018

agente para resolver um problema que vão desde a problemas clássicos de controlo³, como o pêndulo invertido, algoritmos computacionais⁴, como a cópia de palavras, jogos clássicos da Atari⁵, como Pong e a controlo de robôs⁶.

No resto do capítulo serão abordados tópicos relacionados com aprendizagem supervisionada e aprendizagem por reforço, aquelas que são importantes no contexto desta dissertação. Na secção 3.1 é descrito mais pormenorizadamente SL e duas técnicas de SL, Redes Neurais Artificiais (ANNs) e Redes Neurais Profundas (DNNs). Na secção 3.2 é descrito mais pormenorizadamente RL e algumas técnicas de RL, como *Markov Decision Process* (MDP), *Q Learning*, *Deep Q Network* (DQN), *Double Deep Q Learning* (DDQN) e *Deep Deterministic Policy Gradient* (DDPG). Finalmente, são apresentadas algumas implementações destas técnicas em condução autónoma (secção 3.3) e algumas bibliotecas utilizadas para o desenvolvimento destes algoritmos e usadas nesta dissertação (secção 3.4).

3.1 APRENDIZAGEM SUPERVISIONADA

Como referido na introdução ao capítulo 3, aprendizagem supervisionada, ou *Supervised Learning* (SL), é um ramo de ML em que consiste na aproximação de um modelo a um conjunto de dados (*dataset*), com exemplos etiquetados, e depois, prever o significado para novos exemplos. Existem duas categorias de problemas que SL pode resolver, classificação e regressão.

Na classificação, o objetivo é indicar qual ou quais classes pré-definidas pertence um determinado exemplo. Um exemplo é a classificação de dígitos a partir de imagens. Para tal, é necessário utilizar um *dataset* com imagens de dígitos e seu significado, por exemplo o *dataset* MNIST, referenciado nessa introdução.

Na regressão, o objetivo é aproximar uma função dadas as variáveis de entrada para um ou mais valores de saída num espaço contínuo. Um exemplo é, dado um dado número de características de um carro, prever o consumo deste. Para treinar o modelo, pode-se utilizar o *dataset* *Auto MPG Data Set*⁷.

Exemplos de algoritmos de SL são *Naive Bayes*, *K Nearest Neighbour* (KNN), *Support Vector Machine* (SVM), *Random Forest* e ANN. Na secção 3.1.1 e 3.1.2, são descritos os algoritmos mais pertinentes para os objetivos desta dissertação, a ANN e a DNN, um ramo dentro da ANN.

3.1.1 Redes Neurais Artificiais

ANNs, introduzidas por McCulloch e Pitts[12] são estruturas inspiradas nas redes neuronais biológicas, presentes nos cérebros animais. Estas estruturas são utilizadas normalmente como aproximadores de funções difíceis de descrever numa equação. Normalmente, estas estruturas são compostas por camadas de nós, comparados a neurónios num cérebro animal, que estão

³https://gym.openai.com/envs/#classic_control. Acedido em 16/05/2018

⁴<https://gym.openai.com/envs/#algorithmic>. Acedido em 16/05/2018

⁵<https://gym.openai.com/envs/#atari>. Acedido em 16/05/2018

⁶<https://gym.openai.com/envs/#mujoco>. Acedido em 16/05/2018

⁷<https://archive.ics.uci.edu/ml/datasets/Auto+MPG>. Acedido em 25/06/2018

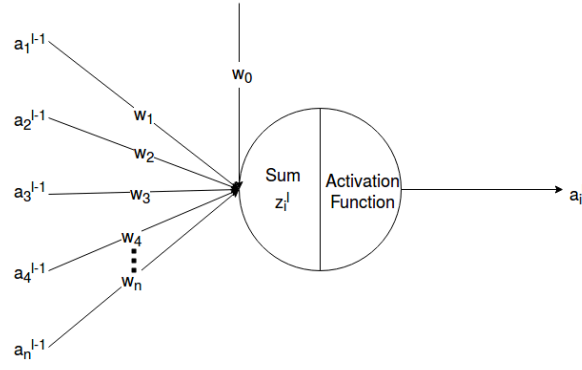


Figura 3.1: Nó de uma ANN

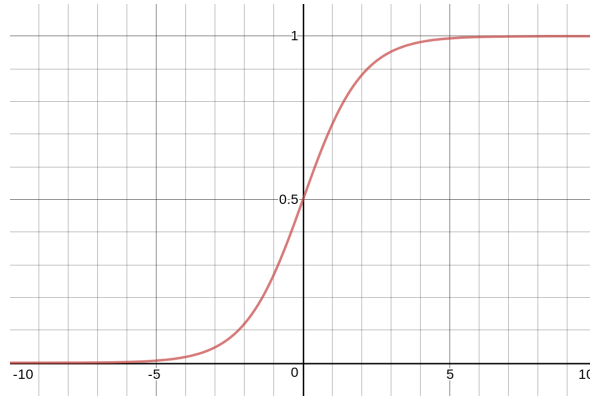


Figura 3.2: Gráfico da função sigmóide

ligados a nós de outras camadas, ligações estas comparadas com as sinapses. Estes nós recebem de camadas anteriores, os valores produzidos pelos seus nós e produzem um novo valor.

Como se pode ver na figura 3.1, o nó i multiplica os valores recebidos, a_1^{l-1} a a_n^{l-1} , pelos respetivos pesos, w_1 a w_n . O valor de w_0 é um valor de *bias*, usado para prevenir que a_i^l seja igual a 0 quando todos os valores de entrada, sejam nulos. Estes valores pesados são somados (equação 3.1) e é aplicada numa função de ativação nesta soma, por exemplo, uma sigmóide (equação 3.3, figura 3.2), que produz o valor de saída a_i^l (equação 3.2).

$$z_i^l = w_j a_j^{l-1} + w_0 \quad (3.1)$$

$$a_i^l = \sigma(z_i^l) \quad (3.2)$$

$$f(x) = \frac{1}{1 + e^{-\theta x}} \quad (3.3)$$

Uma ANN tradicional é composta por uma camada de entrada, com tamanho igual ao número de características das amostras, zero ou uma camada intermédia, com um tamanho definido, e uma camada de saída, com tamanho igual ao número de classes ou valores de saída de uma regressão. Na figura 3.3, está representada uma arquitetura de uma ANN.

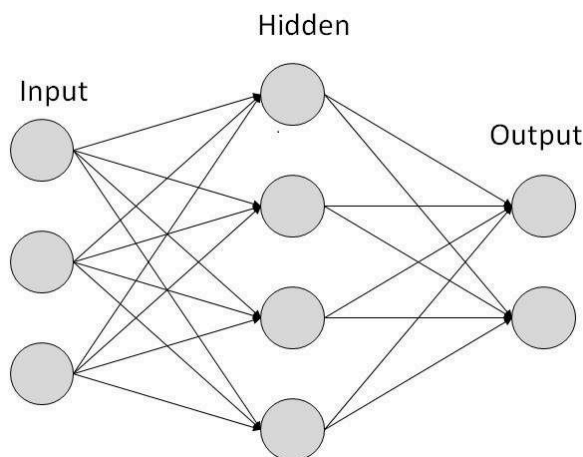


Figura 3.3: Arquitetura de uma ANN. Fonte: ⁸

Para que a ANN possa devolver valores próximos de uma dada função, é necessário que esta estrutura passe por um processo de treino, realizado por um processo de otimização (secção 3.1.1), que, através de retropropagação, procura diminuir o valor de uma função de custo.

Função de Custo

Função de custo é uma medida de erro entre os valores calculados pela ANN e o valor verdadeiro. Esta função é usada na fase de treino, pelo algoritmo de retropropagação, ou *backpropagation*, de maneira a quantificar o erro e ajustar os pesos dos nós. Duas das funções mais usadas como função de custo são *Mean Squared Error* (MSE) (equação 3.4) e *Cross-Entropy* (equação 3.5).

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - h_{\theta}(x_i))^2 \quad (3.4)$$

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^N (y \ln(h_{\theta}(x_i)) + (1 - y) \ln(1 - h_{\theta}(x_i))) \quad (3.5)$$

onde $h_{\theta}(x_i)$ é o valor de saída da ANN para o conjunto de valores x_i .

A segunda função é mais usada para classificação, onde os valores de saída da ANN são $y \in \{0, 1\}$, pois o gradiente para estes valores, quando a função de ativação usada é, por exemplo, a sigmóide, é muito próximo de 0, e se usarmos a primeira função como função de ativação, o algoritmo de otimização converge mais lentamente. No entanto, quando os valores de saída da ANN são valores reais, a primeira função é mais usada.

Backpropagation

Backpropagation [13] é um método usado para calcular e propagar o gradiente da função de custo, necessário para o ajustamento dos parâmetros (pesos e *bias* dos nós) das ANNs.

⁸https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm. Acedido em 21/02/2018

Para tal, um exemplo é apresentado à ANN, que calcula o suposto valor de saída. Este resultado é comparado com o valor esperado usando a função de custo. O erro é calculado para cada nó de saída da ANN. De seguida, estes erros são propagados para as camadas anteriores, até que cada nó tenha o valor do erro associado a si que reflete no resultado final da ANN. Depois, estes erros são usados por uma função de otimização, de maneira a atualizar os parâmetros dos nós e reduzir o erro. Assim, o algoritmo de *backpropagation* para um exemplo é o seguinte (usando como função de ativação a sigmóide), retirado do livro [14].

1. **Exemplo x :** Fazer corresponder a ativação a^1 da camada de entrada ao exemplo dado.
2. **Feedforward:** Para cada camada da rede $l = 2, 3, \dots, L$ calcular o vetor das somas pesadas $z^l = wa^{l-1} + w_0$ e a função de ativação $a^l = \sigma(z^l)$.
3. **Erro de saída δ^L :** Calcular o vetor dos erros $\delta^L = \nabla_a J \odot \sigma'(z^L)$ (\odot : multiplicação elemento a elemento).
4. **Propagar o erro para as camadas a montante:** Para cada camada $l = L-1, L-2, \dots, 2$ calcular $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Resultado:** O gradiente da função custo é dado por $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Otimizadores

Os otimizadores são algoritmos que permitem ajustar os parâmetros de maneira a minimizar a função custo. Exemplos de otimizadores são *Gradient Descent* (GD) e suas variantes, Momentum, Adagrad, Adadelta, e *Adaptive Moment Estimation* (Adam). Estes otimizadores são apresentados com base na descrição do artigo [15].

Gradient Descent. O otimizador GD é um dos primeiros otimizadores e um dos mais populares e dos mais usados. Este otimizador atualiza os parâmetros na direção oposta do gradiente da função custo, minimizando-a. Existem três variantes deste otimizador, dependendo da quantidade de exemplos usados, *Batch Gradient Descent* (BGD), *Stochastic Gradient Descent* (SGD) e *Mini-batch Gradient Descent* (MBGD).

BGD, ou simples GD, calcula o gradiente da função custo para o conjunto total do *dataset* de treino.

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} J(\theta_t) \quad (3.6)$$

onde θ_t é o conjunto dos pesos da ANN no instante t e η é a taxa de aprendizagem, utilizada para controlar a rapidez da aprendizagem.

Como se pode observar na equação 3.6, os parâmetros são atualizados subtraindo o gradiente da função custo multiplicando por η , a taxa de aprendizagem (*learning rate*).

Visto que o gradiente é calculado para o *dataset* inteiro, este cálculo pode ser lento, inviável para grandes *datasets*, que não podem ser totalmente carregados para memória, e não pode ser usado *online*, ou seja, durante a recolha de novos exemplos.

Por outro lado, SGD calcula o gradiente para cada exemplo $x^{(i)}$ e $y^{(i)}$.

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} J(\theta_t; x^{(i)}; y^{(i)}) \quad (3.7)$$

Assim sendo, o cálculo é mais rápido, porque o gradiente não é calculado para todos os exemplos do *dataset*, como também é possível usar *online*.

SGD tem flutuações maiores em relação a BGD, o que permite ir para novos (e possivelmente melhores) mínimos locais, mas pode dificultar a convergência para o mínimo global. Este problema pode ser resolvido reduzindo gradualmente η no processo de aprendizagem.

Entre estes dois subtipos de GD, existe MBGD que calcula o gradiente para um conjunto de n exemplos, ou *batch*.

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} J(\theta_t; x^{(i:i+n)}; y^{(i:i+n)}) \quad (3.8)$$

Este algoritmo reduz a flutuação de SGD, o que resulta numa convergência mais estável. MBGD é um dos algoritmos mais usados para o treino de ANNs e o termo SGD é normalmente usado quando são usados *batches*.

No entanto, [15] enumera vários problemas que estes algoritmos levantam:

- Se a taxa de aprendizagem for demasiado baixa, pode levar a uma convergência muito lenta, enquanto se for muito alta, pode levar a flutuações em torno de um mínimo ou até a divergência.
- A taxa de aprendizagem é aplicada a todos os parâmetros da ANN. Se os dados forem esparsos, os parâmetros não devem ser atualizados da mesma forma entre si, devendo ser efetuadas grandes atualizações em características raras.
- Em [16] é mostrado que o algoritmo pode ficar preso não só em mínimos locais, como também em *saddle points*, ou seja, pontos onde uma dimensão aumenta e outra diminui, e normalmente estão rodeados por planaltos, o que torna difícil para estes algoritmos progredirem, visto que o gradiente é próximo de zero em todas as dimensões.

Momentum. Uma das técnicas usadas para combater os problemas levantados pelo GD e suas variantes é o Momentum [17]. Este método ajuda SGD a localizar mais rapidamente o mínimo global, adicionando uma parte do passo anterior ao passo atual.

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta \nabla_{\theta} J(\theta_t) \\ \theta_t &= \theta_{t-1} - v_{t-1} \end{aligned} \quad (3.9)$$

Para explicar o que sucede, Ruder [15] faz a analogia a uma bola situada numa rampa. A bola acelera quando está a descer e abrandar quando sobe. O mesmo acontece com as atualizações dos parâmetros. O termo γv_t aumenta para as dimensões cujo sentido do gradiente é o mesmo e reduz as atualizações para as dimensões em que o sentido do gradiente é o oposto. Assim, a função converge mais rapidamente e reduz a oscilação, como se pode ver na figura 3.4.

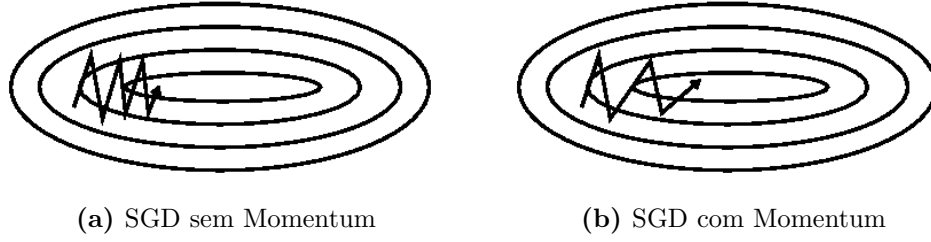


Figura 3.4: Momentum. Fonte: ⁹

Adagrad. Adagrad [18] é um algoritmo baseado no SGD que modifica individualmente a taxa de aprendizagem para cada parâmetro da ANN i . A equação 3.10 representa a atualização para cada parâmetro i utilizando SGD.

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i} \quad (3.10)$$

$$g_{t,i} = \nabla_{\theta_t} J(\theta_t) \quad (3.11)$$

Adagrad modifica a taxa de aprendizagem a cada passo para cada parâmetro baseando no gradiente anterior:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \epsilon} g_{t,i} \quad (3.12)$$

onde $G_t \in \mathbb{R}^{d \times d}$ é uma matriz diagonal cujos elementos da diagonal são a soma dos quadrados dos gradientes no passo t , e ϵ é um termo que evita a divisão por zero. Vetorizando g_t obtém-se:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \odot g_t \quad (3.13)$$

Uma das principais vantagens de Adagrad é a eliminação da necessidade de ajustar a taxa de aprendizagem. No entanto, o algoritmo acumula o quadrado dos gradientes no denominador, um valor positivo, o que leva a taxa de aprendizagem a valores muito pequenos, o que impede a atualização dos parâmetros.

Adadelata. Para combater a grande redução da taxa de aprendizagem pelo algoritmo Adagrad, Adadelata [19] restringe o número de gradientes usados.

Em vez de guardar o histórico dos gradientes, o cálculo das suas somas é definido pela média com decadência (*decaying average*) dos quadrados dos gradientes:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2 \quad (3.14)$$

Substituindo a soma dos quadrados dos gradientes por $E[g^2]_t$:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t} + \epsilon} g_t \quad (3.15)$$

⁹<https://www.willamette.edu/~gorr/classes/cs449/momrate.html>. Acedido em 12/03/2018

Como o denominador é a raiz quadrada da média dos quadrados dos gradientes (*Root Mean Squared* (RMS)) pode-se abreviar a equação para:

$$\theta_{t+1} = \theta_t - \frac{\eta}{RMS[g]_t} g_t \quad (3.16)$$

Zeiler [19] refere que as unidades desta atualização não coincidem. Para tal, é definida outra média com decaência, desta vez com o quadrado das atualizações dos parâmetros:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma) \Delta\theta_t^2 \quad (3.17)$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \quad (3.18)$$

Visto que $RMS[\Delta\theta]_t$ é desconhecido, aproxima-se ao passo anterior $t - 1$, ou seja $RMS[\Delta\theta]_{t-1}$. Substituindo η por $RMS[\Delta\theta]_{t-1}$ obtém-se:

$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \quad (3.19)$$

Ou seja, não é necessário escolher uma taxa de aprendizagem para Adadelta.

Adam. Adam [20] é outro algoritmo que calcula a taxa de aprendizagem para cada parâmetro. Para além de calcular a média com decaência dos quadrados dos gradientes v_t como Adadelta, também calcula a média com decaência dos gradientes m_t como o Momentum:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \quad (3.20)$$

Como os vetores m_t e v_t são inicializados a zero, Kingma e Ba [20] observaram que estes vetores tendem para zero, principalmente durante os primeiros passos, e quando as taxas descendentes são pequenas (ou seja, β_1 e β_2 estão próximos de um).

Para contrariar esta tendência, é aplicada uma correção:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \quad (3.21)$$

Aplicando estes vetores na atualização dos parâmetros:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (3.22)$$

Kingma e Ba [20] mostraram empiricamente que este algoritmo funciona melhor que outros otimizadores de aprendizagem adaptativa, como o Adagrad.

3.1.2 Redes Neurais Profundas

DNNs são ANNs com mais do que uma camada intermédia. No entanto, com o aumento do número de camadas intermédias, o número de nós também aumenta, levando a modelos muito complexos, com milhares, ou mesmo milhões de parâmetros a ajustar. O grande número de camadas intermédias torna estas redes propensas à ocorrência de desaparecimento do gradiente e de *overfitting*.

Desaparecimento do gradiente

Este problema surge devido à diminuição do gradiente no método de *backpropagation*, podendo não chegar às camadas mais longe da camada de saída, tornando impossível de treinar estas camadas. Este problema pode ser resolvido usando, por exemplo, a função *Rectified Linear Unit* (ReLU) [21] como função de ativação (equação 3.23, figura 3.5).

$$f(x) = \max(0, x) \quad (3.23)$$

Outra função de ativação utilizada é a *Exponential Linear Unit* (ELU)[22] (equação 3.24, figura 3.6). Ao comparar os gráficos desta função e da função ReLU, verifica-se que a função ELU permite valores de saída negativos. Isto permite que a média das ativações dos nós da DNN seja um valor próximo de zero, o que pode permitir uma convergência mais rápida.

$$f(x) = \begin{cases} \alpha(e^x - 1) & \text{para } x < 0 \\ x & \text{para } x \geq 0 \end{cases} \quad (3.24)$$

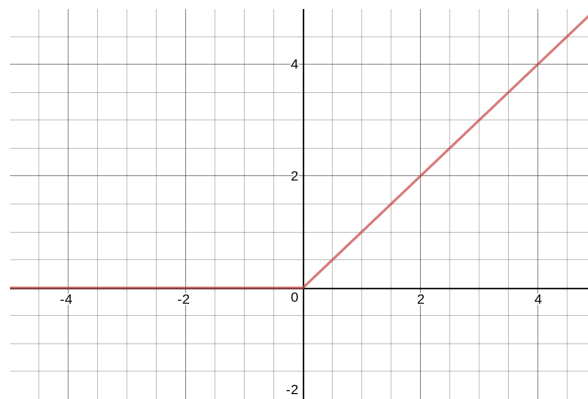


Figura 3.5: Gráfico da função ReLU

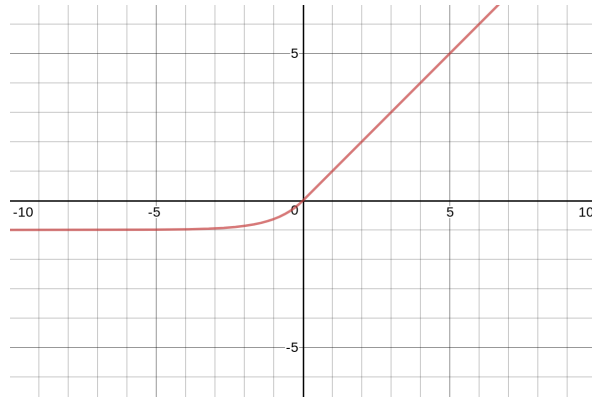


Figura 3.6: Gráfico da função ELU com $\alpha = 1$

Overfitting

Overfitting ocorre quando um dado modelo se especializa em demasia nos dados de treino, que podem conter exemplos que, dadas as suas características, podiam pertencer a outra classe, podendo posteriormente classificar incorretamente exemplos que não pertencem a este conjunto.

Observando a figura 3.7, a linha preta representa a fronteira de um modelo ideal, ou seja, um modelo geral e a linha verde representa a fronteira de um modelo que sofreu *overfitting*.

Quanto mais complexo for o modelo, por exemplo, adicionando mais camadas intermédias a uma DNN, mais suscetível será a *overfitting*.

Uma das técnicas usadas para combater este problema é o *dropout* [23], [24], que treina apenas alguns nós escolhidos aleatoriamente em cada passo (figura 3.8). Depois do treino, todos os nós estão ativos.

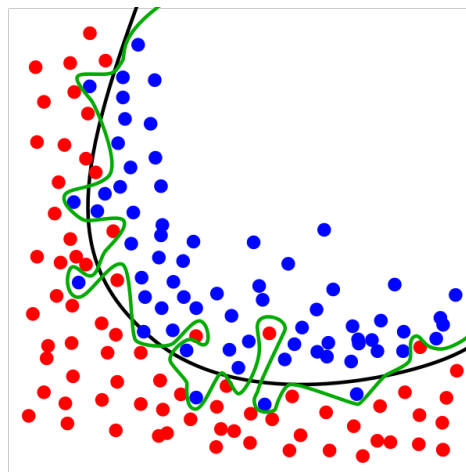


Figura 3.7: Exemplo de *Overfitting*. A linha preta representa a fronteira de um modelo geral e a linha verde representa a fronteira de um modelo que sofreu *Overfitting*. Fonte: ¹⁰

¹⁰<https://en.wikipedia.org/wiki/Overfitting>. Acedido em 12/03/2018

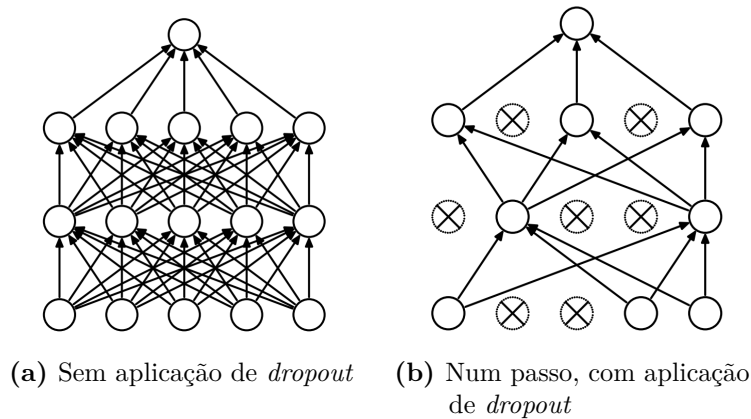


Figura 3.8: Aplicação de *dropout*. Fonte: [24]

Redes Neurais Convolucionais

A Rede Neuronal Convolutiva (CNN), introduzida por Le Cun et al. [3], é um subtipo de DNN, inspirada na percepção visual. A CNN é uma rede neuronal muito poderosa em reconhecimento visual, porque a extração de características é feita pela própria rede, que é treinada com esta, evitando o uso de algoritmos tradicionais de Visão Computacional (VC). Assim, este tipo de DNN pode ser dividida em duas partes, o extrator de características, que pode ser composto por camadas de convolução e de redução, e o classificador, composto por camadas totalmente ligadas, como numa ANN. Exemplos de CNN são a LeNet [3], a AlexNet [4] a VGGNet [5] e para condução autónoma, a PilotNet [1], [25].

As camadas de convolução (*convolution layers*) geram novas imagens, designadas por *feature maps*. Estas novas imagens acentuam as características únicas da imagem original. Cada píxel das novas imagens resulta de uma convolução entre o conjunto constituído pelo píxel correspondente e os píxeis vizinhos da imagem original e um filtro. A operação de convolução pode ser comparada a operações tradicionais de Computação Visual (CV), como *smoothing*, *sharpening* e deteção de arestas. A figura 3.9 mostra como esta operação funciona. O número, o tamanho e o deslocamento do filtro na imagem (na figura 3.9 é representado um filtro com tamanho 2x2 com deslocamento de 1x1, ou seja, os deslocamentos horizontais e verticais são de um píxel) pode ser definido pelo programador.

As camadas de redução (*pooling layers*) reduzem o tamanho dos *feature maps*, reduzindo o número de pesos da rede neuronal. Esta operação resume uma vizinhança de píxeis num único píxel. Esta redução pode ser feita usando, por exemplo, o valor máximo da vizinhança ou a média dos valores da vizinhança. Esta operação é muito similar à operação de convolução. A diferença é que a operação de convolução pode sobrepor os píxeis nos vários cálculos e a camada de redução não. A figura 3.10 mostra um exemplo de uma operação de redução com o valor máximo.

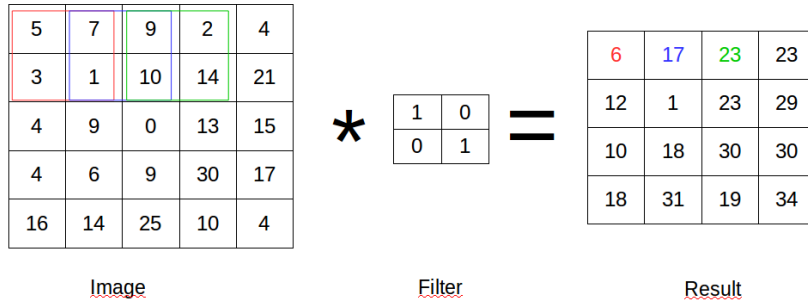


Figura 3.9: Operação de Convolução. O filtro desliza ao longo da imagem original e, para cada posição, é realizada uma soma de produtos.

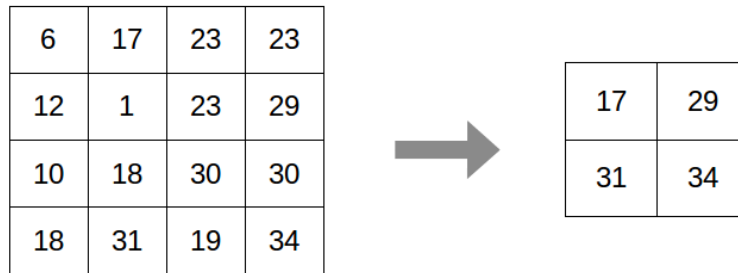


Figura 3.10: Operação de Redução. Neste exemplo, cada quadrado de 4 pixels é transformado num único píxel, cujo valor é o máximo.

3.2 APRENDIZAGEM POR REFORÇO

Aprendizagem por Reforço (ou *Reinforcement Learning* (RL)) é um ramo de ML que não utiliza um *dataset* construído *a priori*, ao contrário de SL. Os métodos de RL consistem na aprendizagem de um ou mais agentes, que sem conhecerem inicialmente o ambiente e a ação a que devem tomar em cada estado s , interagem com o ambiente, levando a transições de estado. O(s) agente(s) é(são) depois recompensado(s) ou penalizado(s), se as ações e as transições de estado aproximam ou afastam o(s) agente(s) do objetivo. A cada conjunto de ação tomada e de transição de estados dá-se um novo passo. Um conjunto de passos cronologicamente ordenados denomina-se por episódio. É através da maximização das recompensas recebidas que o(s) agente(s) aprende(m) uma política π de maneira a agir para alcançar o objetivo. Esta política pode ser extraída de uma função ação-valor V , que avalia o quão boa é uma ação num determinado estado. Esta ação pode depender do estado e da ação tomada nesse estado ou só do estado. Esta função permite determinar de forma indireta a melhor ação a tomar num determinado estado. A figura 3.11 representa uma visão geral da aprendizagem do agente.

De entre os vários algoritmos de RL existentes, ir-se-ão abordar os seguintes: MDP (secção 3.2.1), *Q Learning* (secção 3.2.2), DQN (secção 3.2.3), DDQN (secção 3.2.4) e DDPG (secção 3.2.5).

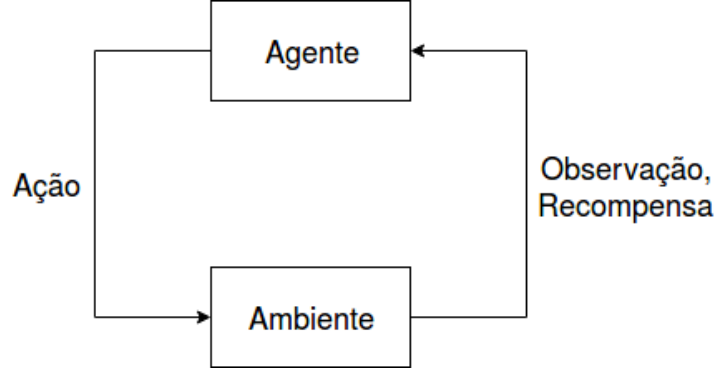


Figura 3.11: *Reinforcement Learning*: Visão Geral

3.2.1 Markov Decision Process

MDP é um formalismo que permite modelar processos de decisão de ações num ambiente totalmente observável. Estes modelos são utilizados em problemas de otimização, resolvidos através de programação dinâmica ou RL.

Em cada instante t , o processo encontra-se no estado s , no qual o agente toma uma ação a . Esta ação leva o processo a transitar para o estado s' , com uma probabilidade $P_a(s, s')$. A decisão é recompensada através de $R(s, a, s')$. Esta recompensa terá importância em recompensas futuras com um fator de desconto $\gamma \in [0, 1]$.

Assim, MDP pode ser definido através do tuplo (S, A, P, R, γ) onde:

- S é um conjunto finito de estados;
- A é um conjunto finito de ações;
- $P_a(s, s')$ é a probabilidade da transição do estado s para o estado s' , tomando a ação a ;
- $R(s, a, s')$ é a recompensa imediata recebida depois da transição do estado s para o estado s' , tomando a ação a ;
- $\gamma \in [0, 1]$ é o fator de desconto, que representa o decaimento da importância entre as recompensas futuras e as recompensas atuais.

O objetivo deste problema é o de encontrar uma política $\pi(s)$ para que o agente possa escolher uma ação a quando estiver no estado s . Esta política (equação 3.25) procura maximizar as recompensas recebidas no imediato e no futuro, calculado através da função valor (equação 3.27).

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_a(s, s') V^*(s') \quad (3.25)$$

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad (3.26)$$

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{\pi(s)}(s, s') V^{\pi}(s') \quad (3.27)$$

3.2.2 *Q Learning*

Q Learning foi apresentado por Watkins[26]. Neste algoritmo, os valores da função ação-valor, neste caso denominada como função Q , são guardados numa tabela, para cada par estado-ação, cujos valores se aproximam aos valores corretos dessa função. A ação escolhida em cada estado (política) corresponde ao maior valor da tabela para o par estado-ação (equação 3.28).

$$\pi(s) = \arg \max_a [Q(s, a)] \quad (3.28)$$

Os valores desta tabela são atualizados a partir da equação 3.29.

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R + \gamma \max_a Q(s', a)] \quad (3.29)$$

onde s é o estado atual, a a ação tomada, R a recompensa recebida pela ação a no estado s , s' o estado seguinte, α é a taxa de aprendizagem e γ um fator de desconto que desvaloriza recompensas futuras.

Para se encontrar possíveis melhores ações do que as escolhidas pela política, aplica-se uma técnica de exploração, como por exemplo ϵ -greedy. Nesta técnica, escolhe-se uma ação aleatória com uma probabilidade ϵ , inicialmente igual a 1. Esta probabilidade diminui ao longo dos passos de aprendizagem a uma taxa definida até a um mínimo maior que 0, de maneira a ser sempre possível explorar.

Para obter a tabela da função Q , utiliza-se o seguinte algoritmo de treino:

```

Inicializar  $Q(s,a)$  arbitrariamente;
enquanto  $Q$  não convergir faça
|   Inicializar  $s$ ;
|   para cada passo do episódio faça
|   |   Escolher  $a$  a partir de  $s$  usando a política derivada de  $Q$  (ex:  $\epsilon$ -greedy) ;
|   |   Aplicar ação  $a$ ;
|   |   Observar  $r, s'$ ;
|   |    $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_a Q(s', a)]$ ;
|   |    $s \leftarrow s'$ ;
|   |   se  $s$  for terminal então
|   |   |   parar
|   |   fim
|   fim
fim

```

Algoritmo 1: *Q Learning*

3.2.3 *Deep Q Network*

Q Learning (secção 3.2.2) utiliza uma tabela para representar a função Q . No entanto, torna-se impraticável guardar em memória essa tabela para um problema com um grande número de estados e/ou ações. Para reduzir a utilização de memória, DQN[27] substitui a tabela por uma ANN, onde a entrada é o estado atual e as saídas são os valores da função Q

para cada ação. Esta ideia já foi utilizada pelo algoritmo *Neural Fitted Q Iteration* (NFQ) [28]. Para treinar esta ANN, utiliza-se a equação 3.29, com $\alpha = 1$, como alvo:

$$Q(s, a, \theta) \leftarrow R + \gamma \max_a Q(s', a, \theta) \quad (3.30)$$

onde θ é o conjunto de parâmetros da ANN.

No entanto, a ANN diverge devido a correlação entre os vários passos e a valores alvo não estacionários. Para combater estes dois problemas, são aplicadas duas técnicas, *experience replay* e a utilização de outra ANN chamada *target network*, idêntica à primeira.

Na técnica *experience replay*, é guardado numa memória com uma capacidade máxima definida os tuplos de cada passo (s, a, r, s' , s é terminal?). Na fase de treino da ANN, é retirado um conjunto de tuplos e o treino é feito com estes e não só com o tuplo do passo atual.

A *target network* é utilizada para o cálculo do valor alvo utilizado na equação 3.30, obtendo:

$$Q(s, a, \theta) \leftarrow R + \gamma \max_a Q(s', a, \theta') \quad (3.31)$$

onde θ' é o conjunto de parâmetros da *target network*. A atualização destes parâmetros é feita de C em C passos, definido pelo programador.

Assim, o algoritmo de treino da DQN é o seguinte:

```

Inicializar memória D com capacidade N;
Inicializar θ arbitrariamente;
 $\theta' \leftarrow \theta$ ;
enquanto Q não convergir faça
    Inicializar s;
    para cada passo do episódio faça
        Escolher a a partir de s usando a política derivada de Q (ex:  $\epsilon$ -greedy) ;
        Aplicar ação a;
        Observar r, s';
        Guardar em D o tuplo (s, a, r, s', s terminal?);
        Retirar de D um conjunto aleatório de tuplos;
        para cada tuplo j faça
            
$$y_j = \begin{cases} r_j & \text{se } s_j \text{ for terminal} \\ r_j + \gamma \max_{a_j} Q(s'_j, a_j, \theta') & \text{se } s_j \text{ não for terminal} \end{cases};$$

        fim
        Treinar  $\theta$  com y;
        se  $\text{passo} \% C == 0$  então
            |  $\theta' \leftarrow \theta$ 
        fim
         $s \leftarrow s'$ ;
        se s for terminal então
            | parar
        fim
    fim
fim

```

Algoritmo 2: *Deep Q Network*

3.2.4 Double Deep Q Learning

Q learning e DQN tendem a sobrestimar os valores da função ação-valor devido à falta de flexibilidade do aproximador da função ação-valor [29] ou ruído [30], [31]. O algoritmo DDQN [32], versão do algoritmo Double Q Learning [30] que utiliza ANNs como aproximador da função ação-valor, combate este problema, modificando a equação de atualização dos valores da função ação-valor (equação 3.31) para a equação 3.32.

$$Q(s, a, \theta) \leftarrow R + \gamma Q(s', \arg \max_{a'} Q(s', a', \theta), \theta') \quad (3.32)$$

Nesta equação, substitui-se a utilização do mesmo aproximador da função ação-valor para escolher e avaliar a ação, por dois aproximadores. Segundo os autores de [32], se utilizar apenas um aproximador, é mais provável selecionar valores sobrestimados, resultando em estimativas sobreotimistas. Para prevenir esta situação, utiliza-se dois aproximadores, um para a seleção da ação e outro para a avaliação dessa ação.

No algoritmo DDQN, o aproximador que avalia a ação é substituído pela *target network* utilizada no algoritmo DQN, sendo que a ANN principal é o aproximador que escolhe a ação, modificando minimamente o algoritmo DQN, bastando alterar a atualização da função ação-valor.

3.2.5 Deep Deterministic Policy Gradient

Os algoritmos de RL apresentados anteriormente são utilizados para problemas com um espaço de ações discreto. Para que estes algoritmos possam ser usados em problemas cujo espaço de ações seja contínuo, deve-se discretizar este espaço. No entanto, esta discretização pode levar a um grande número de ações, devido ao problema da dimensionalidade, o que pode tornar impossível a convergência dos algoritmos. Para evitar esta discretização, existem outros algoritmos de RL utilizados para problemas cujo espaço de ações seja contínuo. Um destes algoritmos é o DDPG [33]. Este algoritmo é uma abordagem ator-crítico baseado no algoritmo *Deterministic Policy Gradient* (DPG) [34].

Nos algoritmos DPG e DDPG são usadas duas funções, a função ator e a função crítico. A função ator é responsável por definir a política, mapeando os estados para a melhor ação a tomar. A função crítico é responsável por avaliar o quão boa é a ação escolhida pela função ator, ou seja, a função ação-valor. A diferença entre estes dois algoritmos é a representação destas funções. O algoritmo DPG utiliza tabelas para representar estas funções, enquanto que o algoritmo DDPG utiliza ANNs como aproximadores a essas funções.

No algoritmo DDPG, a ANN da função crítico é atualizada segundo a equação 3.33, semelhante à usada no DQN:

$$Q(s, a, \theta^Q) \leftarrow R + \gamma \max_a Q(s', a, \theta^Q) \quad (3.33)$$

$$a = \mu'(s', \theta^{\mu'}) \quad (3.34)$$

onde μ é a função ator.

Para treinar a ANN da função ator, procura-se mover esta na direção ascendente do gradiente dos seus parâmetros usando o gradiente da função crítico (equação 3.35), visto que o objetivo é escolher a melhor ação, escolhendo o máximo da função ação-valor.

$$\frac{\partial J(\theta^\mu)}{\partial \theta^\mu} = \mathbb{E}_{x \sim p(x|\theta^\mu)} \left[\frac{\partial Q(s, a, \theta^Q)}{\partial a} \frac{\partial a}{\partial \theta^\mu} \right] \quad (3.35)$$

Ao contrário dos algoritmos anteriores, as *target networks* do algoritmo DDPG são atualizadas em todos os passos seguindo a equação 3.36.

$$\theta' \leftarrow \tau \theta + (1 - \tau) \theta' \quad (3.36)$$

onde τ é a taxa de atualização da *target network*.

Como em *Q Learning*, DQN e DDQN, aplica-se uma técnica de exploração para se poder encontrar a política ótima. Esta técnica passa por adicionar ruído à ação escolhida pela ANN da função ator (equação 3.37). Os autores de [33] utilizaram para as suas experiências o ruído Ornstein-Uhlenbeck [35]. Esta escolha foi feita pois é gerado ruído correlacionado para a exploração eficiente em problemas de controlo físico com inércia.

$$\mu'(s) = \mu(s|\theta_t^\mu) + \mathcal{N} \quad (3.37)$$

onde \mathcal{N} é o ruído adicionado.

Assim, obtém-se o seguinte algoritmo:

```

Inicializar memória D com capacidade N;
Inicializar  $\theta^\mu$  e  $\theta^Q$  arbitrariamente;
 $\theta^{\mu'} \leftarrow \theta^\mu$ ;
 $\theta^{Q'} \leftarrow \theta^Q$ ;
enquanto  $Q$  e  $\mu$  não convergirem faça
    Escolher novo processo  $\mathcal{N}$ ;
    Obter s inicial;
    para cada passo do episódio faça
        Escolher ação  $a = \mu(s|\theta_t^\mu) + \mathcal{N}$  ;
        Tomar ação a;
        Observar r, s';
        Guardar em D o tuplo (s, a, r, s', terminal);
        Retirar de D um conjunto aleatório de tuplos;
        para cada tuplo  $j$  faça
             $y_j = \begin{cases} r_j & \text{se } s_j \text{ for terminal} \\ r_j + \gamma \max_a Q(s', \mu'(s', \theta^{\mu'}), \theta^{Q'}) & \text{se } s_j \text{ não for terminal} \end{cases}$ ;
        fim
        Treinar  $\theta^Q$  com  $y$ ;
        Treinar  $\theta^\mu$ :  $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a, \theta^Q) \nabla_{\theta^\mu} \mu(s, \theta^\mu)$ ;
         $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$ ;
         $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$ ;
         $s \leftarrow s'$ ;
        se  $s$  for terminal então
            | parar
        fim
    fim
fim

```

Algoritmo 3: *Deep Deterministic Policy Gradient*

3.3 CONDUÇÃO AUTÓNOMA E APRENDIZAGEM AUTOMÁTICA

Nestes últimos anos, têm aparecido cada vez mais veículos autónomos, capazes de se controlar a si próprios em algumas situações como seguir a via onde circula, ultrapassar outros veículos, evitar colisões e estacionar. Normalmente, os agentes para estas operações utilizam técnicas de ML e não de VC, devido à sua complexidade.

Uma das primeiras implementações feitas para a integração de algoritmos de SL na Condução Autónoma foi realizada em 1993, pelo projeto *Autonomous Land Vehicle In a Neural Network* (ALVINN) [36], onde se utilizou uma ANN para mapear uma imagem num ângulo de direção. Mais tarde, [1] utilizou uma CNN, com o mesmo fim. Estas duas implementações pertencem à categoria das abordagens comportamental reativa, ou *behavior reflex approach*, que usam um regressor para mapear diretamente os valores lidos dos sensores em comandos de direção [37].

Na abordagem de percepção mediada, ou *mediated perception approach*, consiste em analisar os dados recolhidos pelos sensores para reconhecimento de objetos relevantes para a condução, como marcações da estrada, sinais de trânsito, semáforos, outros veículos, pedestres, etc. Estes objetos combinam numa representação do mundo que envolve o veículo. Com esta informação, o agente, por exemplo, implementado com uma ANN, toma uma decisão.

Uma terceira abordagem, a abordagem de percepção direta, ou *direct perception approach*, proposta em [37], é retirada informação dos dados recolhidos pelos sensores sobre o ângulo do veículo em relação ao eixo da estrada, distância às marcações da estrada, distância a outros veículos nas vias adjacentes à via onde o próprio veículo circula, etc. Com esta informação, o agente toma uma decisão.

Em relação a implementações com RL, foi construído em [38] um agente implementado com DQN capaz de controlar um veículo no simulador JavaScript Racer¹¹. Este agente recebe como estado a imagem do simulador apresentada ao utilizador e escolhe uma ação num espaço discreto com 9 elementos, correspondentes às combinações do conjunto acelerar, travar e manter velocidade, e do conjunto curvar para a direita, curvar para a esquerda e seguir em frente. Também foi construído em [39] um agente com DQN capaz de controlar um veículo no simulador Vdrift¹². Este agente, para além de receber a imagem do simulador, também recebe como estado a velocidade do veículo, a posição transversal em relação à pista e o ângulo em relação ao eixo da faixa. O agente pode escolher uma ação num espaço discreto com 9 ações, semelhante ao da implementação anterior.

Também foram implementados agentes para espaços de ações contínuos. Em [33], para além da apresentação do algoritmo DDPG, são apresentados alguns agentes para resolver diferentes tarefas, incluindo condução autónoma. Nesta tarefa, o agente controla um veículo no simulador TORCS¹³, recebendo deste, como representação do estado, uma imagem, produzindo uma ação pertencente a um espaço contínuo.

3.4 TECNOLOGIAS

Esta secção descreve algumas tecnologias muito utilizadas para o desenvolvimento de algoritmos de ML. Estas são o TensorFlow (secção 3.4.1), o Keras (secção 3.4.2) e o TFLearn (secção 3.4.3).

3.4.1 TensorFlow

Tensorflow¹⁴[40] é uma biblioteca de código aberto, desenvolvida pela *Google* e sucessora da biblioteca DistBelief [41], desenhada para alto desempenho de cálculo numérico, essencial para a implementação de algoritmos de ML. Esta biblioteca permite que os algoritmos sejam executados facilmente de uma forma distribuída, quer em *Central Processing Units* (CPUs), quer em GPUs.

¹¹<https://github.com/jakesgordon/javascript-racer>. Acedido em 29/05/2018

¹²<http://vdrift.net/>. Acedido em 29/05/2018

¹³<https://sourceforge.net/projects/torcs/>. Acedido em 29/05/2018

¹⁴<https://www.tensorflow.org/>

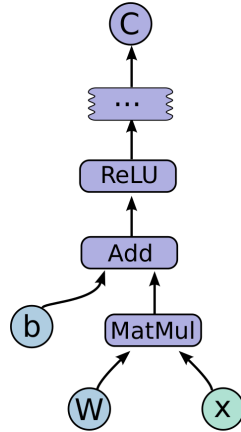


Figura 3.12: Exemplo de um grafo de Tensorflow. Fonte:[40]

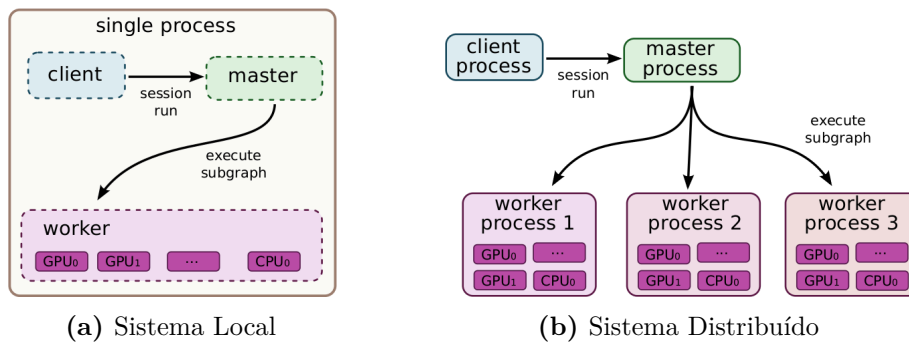


Figura 3.13: Estruturas dos sistemas de TensorFlow. Fonte:[40]

A computação de um algoritmo construído em **Tensorflow** é organizada num grafo (figura 3.12), que representa um fluxo de dados, composto por um conjunto de nós, que representam operações com zero ou mais entradas e zero ou mais saídas. Os arcos do grafo, os fluxos de dados, chamam-se tensores. Também existem outro tipo de arcos, chamados dependências de controlo, que indicam ao nó fonte da dependência de controlo que deve terminar a sua execução antes do nó destino começar a executar. Estes grafos são executados em sessões (**Session**). Para executar a sessão, chama-se a função `run()`, que tem como argumentos um conjunto de saídas que necessitam de ser calculadas e um dicionário com os tensores de entrada e seus valores.

Os principais componentes no sistema do **Tensorflow** são o cliente, que utiliza a interface **Session** para comunicar com o *master* e um ou mais processos *worker*, em que cada um destes processos é responsável pelo acesso a um ou mais dispositivos computacionais (CPUs e GPUs) e pela execução dos nós dos grafos nestes dispositivos, instruídos pelo *master*. Estes componentes podem estar na mesma máquina, num sistema local (figura 3.13a), ou em máquinas diferentes, num sistema distribuído (figura 3.13b).

Esta biblioteca está disponível em várias linguagens tais como **Python**, **C++**, **Java**, **JavaScript** e **Go**. Para além de fornecer estas APIs, também fornece ferramentas de visualização como o **TensorBoard**.

Esta biblioteca fornece uma API de baixo nível, o que não é ideal para iniciantes. No

entanto, existem outras bibliotecas que abstraem o **TensorFlow**, como o **Keras** e o **TFLearn**.

3.4.2 Keras

Keras¹⁵ é uma biblioteca de código aberto, que permite desenvolver e treinar ANNs de uma forma fácil e rápida. Esta biblioteca é uma biblioteca de alto nível que pode ser executada sobre outras bibliotecas usadas para o desenvolvimento de algoritmos de ML, tais como **Tensorflow**, **Theano**¹⁶ e **CNTK**¹⁷. Esta biblioteca está disponível em **Python**. Sendo uma biblioteca de alto nível, é menos flexível e poderá ser mais difícil de desenvolver algoritmos mais complexos. No entanto, é possível manipular as variáveis criadas com a biblioteca **Keras** através da biblioteca de baixo nível utilizada, permitindo a criação de modelos mais complexos do que essa biblioteca permite. Todavia, depois desta manipulação, é impossível de mudar de biblioteca de baixo nível.

3.4.3 TFLearn

TFLearn¹⁸ é uma biblioteca de alto nível que abstrai a biblioteca **TensorFlow**, permitindo desenvolver e treinar ANNs de uma forma fácil e rápida. Tal como o **Keras**, é possível manipular as variáveis criadas pela biblioteca **TFLearn**, através da biblioteca de baixo nível, neste caso o **Tensorflow**.

¹⁵<https://keras.io/>. Acedido em 11/04/2018

¹⁶<http://deeplearning.net/software/theano/>. Acedido em 11/04/2018

¹⁷<https://www.microsoft.com/en-us/cognitive-toolkit/>. Acedido em 11/04/2018

¹⁸<http://tflearn.org/>. Acedido em 11/04/2018

Condutor Autónomo

A condução do veículo ROTA é realizada por dois módulos, piloto e copiloto (ver secção 2.2.1). O copiloto é responsável por escolher ao longo do percurso qual é o melhor modo de condução, enquanto que o piloto realiza a condução propriamente dita. Vários comportamentos de condução podem ser estabelecidos: condução baseada em localização global; condução baseada na localização relativa à faixa de rodagem; condução baseada em odometria; etc. A solução existente, baseada na localização na faixa de rodagem, é implementada usando um controlador PD que tenta manter o veículo num dado posicionamento transversal e orientação na faixa. O principal objetivo deste trabalho foi o de reimplementar este módulo usando aprendizagem automática.

Duas abordagens foram seguidas, uma usando aprendizagem supervisionada e outra aprendizagem por reforço. Na primeira abordagem (secção 4.1), o módulo de condução (piloto) é implementado por uma CNN, tendo como entrada uma região de interesse da imagem da estrada captada por uma câmara RGB e como saída a velocidade angular a aplicar ao veículo. Na segunda abordagem (secção 4.2), o piloto é implementado por uma rede neuronal, recebendo como entrada estimativas do posicionamento transversal e da orientação do veículo na faixa de rodagem e produzindo a velocidade angular a aplicar.

As redes criadas nestas duas abordagens foram treinadas num ambiente de simulação realista. A CNN implementada na primeira abordagem foi testada e usada em ambiente real na prova de condução autónoma durante o FNR.

Para o desenvolvimento dos algoritmos de ML utilizou-se **TensorFlow** e **Keras**. Estes algoritmos estão integrados em nós ROS para integrarem no sistema de controlo do veículo.

4.1 PILOTO IMPLEMENTADO COM APRENDIZAGEM SUPERVISIONADA

Este capítulo descreve a implementação do piloto utilizando SL, desde a criação do *dataset* (secção 4.1.1) e pré-processamento das imagens (secção 4.1.2), passando pela descrição da arquitetura da ANN utilizada para controlar o ROTA e hiperparâmetros (secção 4.1.3), o

treino da ANN (secção 4.1.4) e a aplicação do modelo no Piloto do ROTA (secção 4.1.5). Finalmente, são mostrados os resultados obtidos (secção 4.1.6).

A rede neuronal utilizada para esta implementação é do tipo CNN (secção 3.1.2). Para entrada desta rede utilizaram-se as imagens captadas pela câmara RGB do veículo ROTA. Esta rede devolve a velocidade angular. O *dataset* utilizado para o treino da rede neuronal foi criado a partir de dados gravados de provas de condução no ambiente de simulação, sendo que o veículo foi controlado manualmente, utilizando um comando remoto, a uma velocidade constante.

O código criado para o aumento do *dataset* é adaptado de ^{1 2 3}. O código de treino tem como base os tutorias do Keras ⁴. A integração no controlador do veículo foi feita de raiz.

4.1.1 Construção do *Dataset*

Como referido no início do capítulo, a rede foi treinada com base num *dataset* obtido a partir de informação recolhida do ambiente de simulação. Cada exemplo deste *dataset* é constituído por uma imagem da estrada, que corresponde à amostra, e pela velocidade angular a aplicar ao veículo para o manter na faixa de rodagem, que corresponde à etiqueta. O *dataset* foi construído em duas fases: criação de um *dataset* base diretamente a partir do ambiente de simulação e seu enriquecimento posterior.

Criação do Dataset

Para criar o *dataset*, foi usado o ambiente de simulação. Nele, o veículo ROTA foi conduzido manualmente, usando um comando remoto. A câmara RGB foi colocada com uma inclinação de $\pi/8$ rad, apontando para a estrada. A pista usada foi a mostrada na figura 4.1 e a velocidade linear de condução foi fixada em 0,8 m/s. Utilizando a ferramenta *roslab*, gravaram-se as imagens da câmara e as velocidade angulares aplicadas ao veículo em consequência dos comandos dados pelo utilizador.

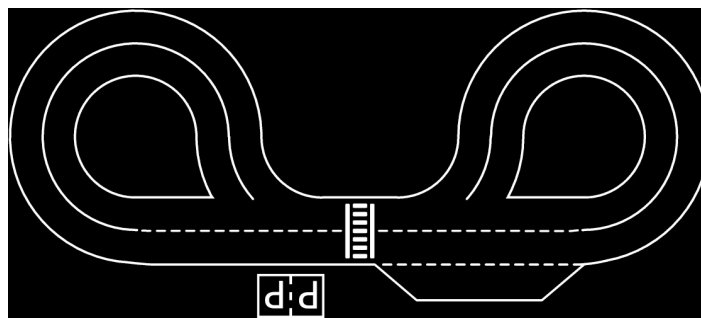


Figura 4.1: Pista usada na criação do *dataset*

¹<https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9>. Acedido em 12/02/2018

²<https://navoshta.com/end-to-end-deep-learning/#data-augmentation>. Acedido em 12/02/2018

³<https://blog.coast.ai/training-a-deep-learning-model-to-steer-a-car-in-99-lines-of-code-ba94e0456e6a>. Acedido em 12/02/2018

⁴<https://keras.io/>. Acedido em 11/04/2018

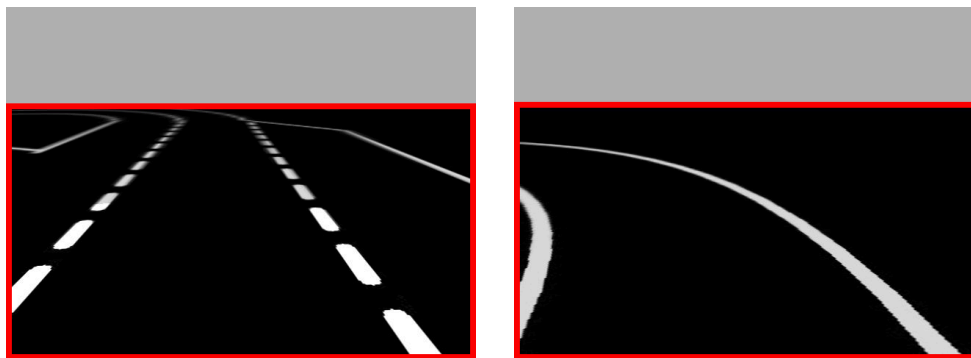


Figura 4.2: Exemplos do *dataset*. A zona interior à moldura vermelha é a zona de interesse extraída.

Os dados gravados foram posteriormente tratados de modo a construir um *dataset* de treino e teste. Atendendo à forma como o ambiente de simulação funciona, nem todas as imagens captadas tem uma velocidade angular associada. Para tratar estes dados, foi necessário criar uma ferramenta que, a partir dos vídeos gravados, extraí-se as imagens, extraíndo só as imagens com uma velocidade angular associada. Nesta extração, as imagens são cortadas na parte superior, o horizonte, que não contém informação útil (figura 4.2). A eliminação desta zona não útil tem dois benefícios, a eliminação de uma fonte de ruído e a redução da dimensão do vetor de entrada. Juntamente com estas imagens, foram também extraídas as velocidades angulares correspondentes. A associação imagem-velocidade angular está presente no nome das imagens e num ficheiro estruturado *CSV*, de maneira a tornar a associação e representação de todo o *dataset* mais fácil.

Com base neste processamento, foi obtido um *dataset* composto por 3985 exemplos.

Enriquecimento do Dataset

Uma forma de combater o *overfitting* (secção 3.1.2) é treinar o modelo com uma grande quantidade de exemplos. No entanto, o tamanho do *dataset* criado não é suficiente, sendo necessário aumentá-lo. Há várias formas de aumentar o *dataset*. A maneira mais direta de aumentar o *dataset* é encontrar novos exemplos, a partir de outros *datasets* ou criando novos exemplos. No entanto, pode não haver mais *datasets* para este caso e é moroso criar mais exemplos. Uma outra maneira de aumentar o número de exemplos é criá-los a partir dos exemplos existentes, transformando-os. Estas transformações podem ser feitas através de transformações tradicionais, como translações, rotações, espelho, transformação da perspetiva, *blur*, adição de ruído, entre outras, até a utilização de técnicas de *Deep Learning*, como *Generative Adversarial Networks* (GANs) [42].

Foram gerados mais exemplos para o *dataset* aplicando-se as seguintes transformações:

- **Adição de sombras:** robustez a sombras;
- **Variação do brilho:** robustez a variações de iluminação;
- **Deslocações verticais:** robustez a vibrações da câmara;
- **Inversões horizontais:** aumentar o número de casos para os valores da velocidade angular. Observando a figura 4.3, que representa o histograma de velocidades angulares

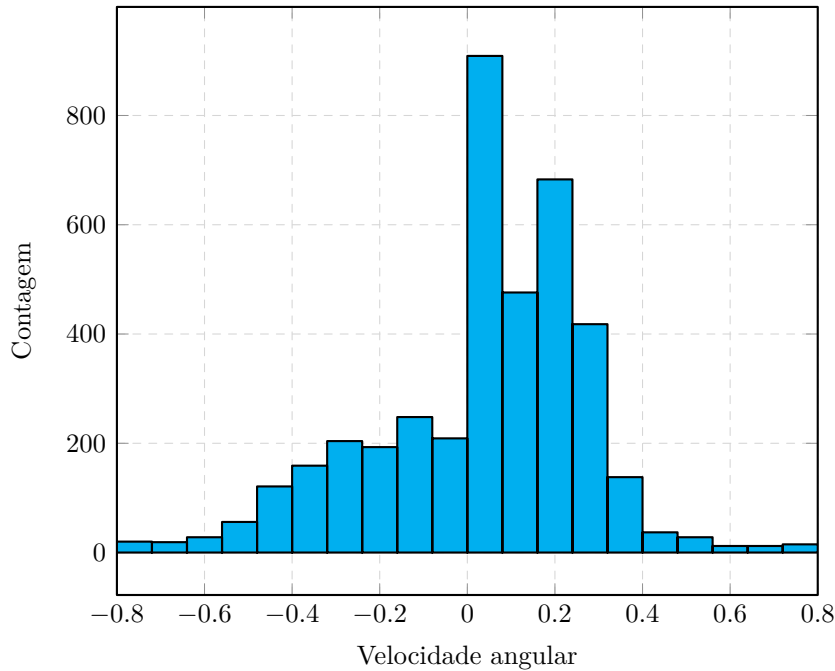


Figura 4.3: Histograma da velocidade angular dos exemplos do *dataset*

dos exemplos no *dataset* base criado, verifica-se que existem mais exemplos com velocidade angular positivas do que negativas. Aplicando esta transformação, consegue-se eliminar esta discrepância.

O valor da etiqueta dos exemplos criados (velocidade angular) através das transformações acima descritas é igual à etiqueta dos exemplos originais correspondentes, à exceção da última transformação (inversões horizontais), cujo valor da etiqueta do exemplo criado é o simétrico do valor da etiqueta do exemplo original correspondente.

4.1.2 Pré-processamento da imagem

Antes da classificação feita pelo modelo da imagem recebida, é necessário realizar um pré-processamento a esta. As transformações são as seguintes, realizadas por ordem:

- **Conversão:** A imagem é recebida no formato da mensagem Image⁵ do ROS. Para que esta possa ser utilizada, é necessária converter para o formato usado pela biblioteca OpenCV, utilizando o módulo `cv_bridge`⁶;
- **Recorte:** Como referido na secção 4.1.1, a imagem é recortada à zona de interesse, ou seja, a estrada;
- **Conversão para escala de cinza:** A imagem é convertida para escala de cinza;
- **Redimensionamento:** A imagem é redimensionada para 180x86, para reduzir o número de píxeis e, consequentemente, o número de entradas da ANN;
- **Normalização:** Os valores dos píxeis são normalizados de [0;255] para [0;1];

⁵http://docs.ros.org/api/sensor_msgs/html/msg/Image.html. Acedido em 28/05/2018

⁶http://wiki.ros.org/cv_bridge. Acedido em 28/05/2018

- **Binarização:** Para utilizar o modelo em outras pistas de outras cores, é feito uma binarização da imagem. Esta operação tem em consideração a possibilidade de construção do *dataset* com base em recolha feita no ambiente real, onde as cores não são perfeitas, ao contrário na recolha feita no ambiente de simulação. Esta operação só é feita na utilização do modelo no ambiente real.

4.1.3 Arquitetura da Rede Neuronal

Como estão a ser utilizadas imagens para calcular o comando de direção do ROTA, a ANN usada é uma CNN. A arquitetura final da rede, adaptada da arquitetura descrita em [1], está representada, da camada de entrada para a camada de saída, na figura 4.4 e com a especificação das camadas na tabela 4.1.

Ao analisar esta arquitetura, repara-se que não existem camadas de redução. A redução das *feature maps* é realizada pelo deslocamento maior que um, nas direções horizontal e vertical, dos filtros das camadas de convolução.

Durante o treino, as camadas totalmente ligadas tiveram um *dropout* (secção 3.1.2) com uma probabilidade de 20% de desativação.

Como o valor do comando de direção é um valor real, entre -1 e 1, utilizou-se como função de ativação da camada de saída a tangente hiperbólica.

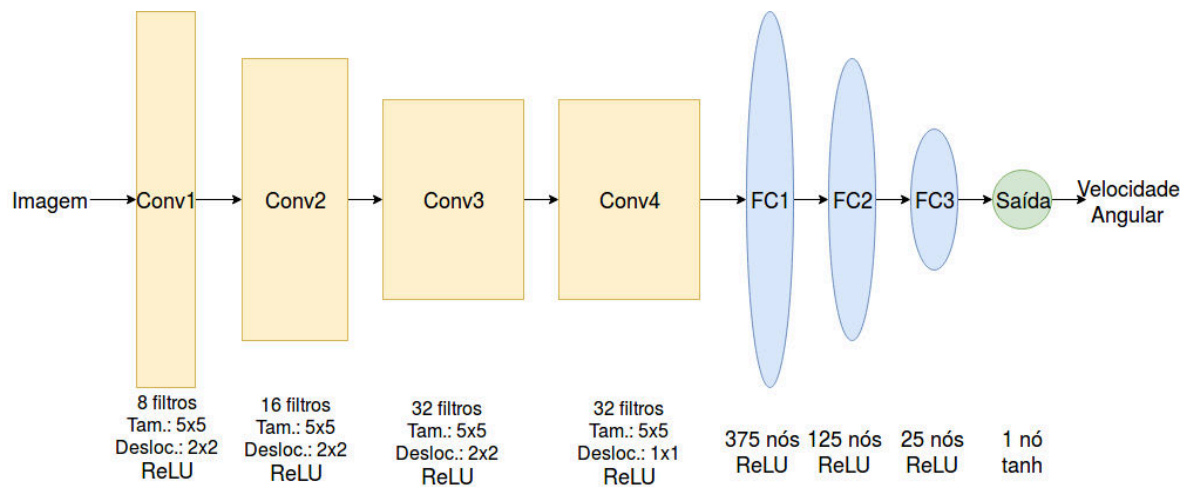


Figura 4.4: Arquitetura da CNN do piloto

Tabela 4.1: Arquitetura da CNN do piloto

Nome da Camada	Tipo de Camada	Nº de nós	Nº de filtros	Tamanho do filtro	Deslocamento do filtro	Função de ativação
Conv1	Convolutacional	-	8	5x5	2x2	ReLU
Conv2	Convolutacional	-	16	5x5	2x2	ReLU
Conv3	Convolutacional	-	32	5x5	2x2	ReLU
Conv4	Convolutacional	-	32	5x5	1x1	ReLU
FC1	Totalmente Ligada	375	-	-	-	ReLU
FC2	Totalmente Ligada	125	-	-	-	ReLU
FC3	Totalmente Ligada	25	-	-	-	ReLU
Saída	Totalmente Ligada	1	-	-	-	tanh

4.1.4 Treino

Para treinar o modelo definido na secção 4.1.3, dividiu-se o *dataset* em duas partes. A primeira parte, composta por 80% do *dataset*, é usada para o treino da CNN, chamado *dataset* de treino. A segunda parte, composta pelos restantes 20% do *dataset*, é usada para validar a CNN depois do treino, chamado *dataset* de validação. As imagens pertencentes ao *dataset* de treino, antes de serem enviadas para a CNN, são pré-processadas com as operações descritas na secção 4.1.1 e em 4.1.2. A rede é treinada vinte e cinco vezes (passos) com o conjunto de treino, dividido em conjuntos de trinta e duas imagens. No início de cada passo, as imagens são baralhadas para remover correlações entre os conjuntos de imagens. Para treinar a CNN, é usada a função MSE como função de custo e para reduzir o valor desta função é utilizado o otimizador Adam (secção 3.1.1) com uma taxa de aprendizagem de $1 * 10^{-4}$.

4.1.5 Utilização do Modelo

Após o treino (secção 4.1.4), o modelo obtido é incorporado num novo nó ROS que controla o ROTA utilizando a ANN para conduzir. No início, este nó coloca a câmara na posição utilizada na criação do *dataset* (secção 4.1.1), ou seja, $\pi/8$ rad. De seguida, o nó subscreve o tópico de imagens da câmara. Cada imagem recebida é pré-processada (secção 4.1.2), publicada num novo tópico, para efeitos de teste e/ou depuração, e enviada para a CNN. No final, é enviado um comando de movimento composto por uma velocidade constante, neste caso 1.0 m/s, e uma velocidade angular, o resultado da CNN.

4.1.6 Resultados

Após o treino, obtiveram-se os gráficos da figura 4.5, que representam o valor da função custo com o *dataset* de treino e teste ao longo das iterações para todos os exemplos dos *datasets*. As linhas mais opacas representam valores suavizados e as linhas mais transparentes representam os valores recolhidos.

Ao analisar os gráficos, ambos decrescem ao longo das iterações, o que significa que o modelo convergiu para a solução. Verifica-se também que ao longo das iterações, o valor da função de custo para o *dataset* de teste é inferior ao valor para o *dataset* de treino, exceto nas últimas iterações, em que é ligeiramente superior. Isto indica que o modelo não sofreu *overfitting*.

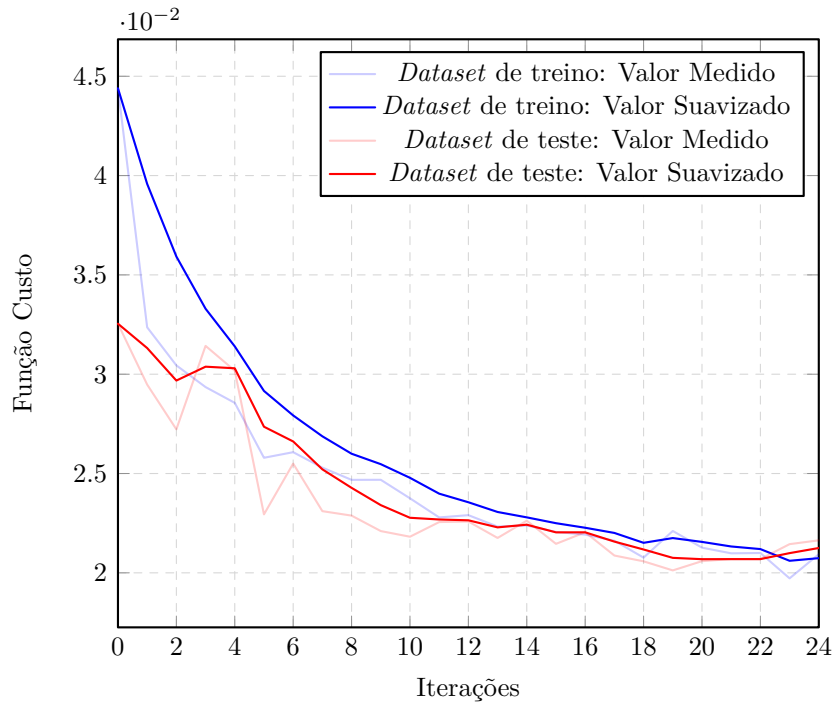


Figura 4.5: Gráficos da Função Custo do Treino do Modelo SL. A suavização corresponde a uma soma entre 0.6 do valor anterior suavizado e 0.4 do valor atual medido

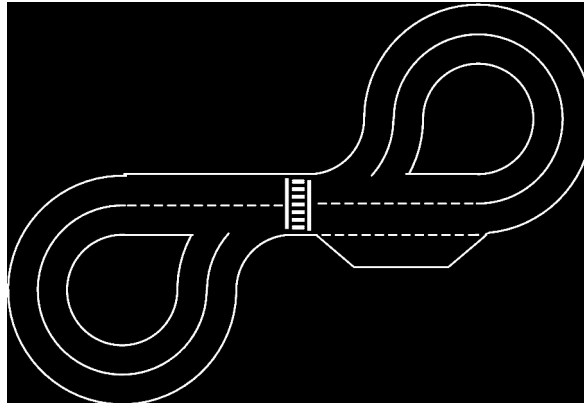


Figura 4.6: Pista de Teste

Após aplicado o modelo no piloto do ROTA, testou-se não só na pista onde foi construído o *dataset*, como também numa outra pista, representada na figura 4.6.

Os vídeos nas hiperligações ⁷ e ⁸ mostram este piloto a conduzir pelas pistas de treino e de teste, respetivamente.

Condução em Pista Real. Na Prova de Condução Autónoma no FNR de 2018, experimentou-se usar o piloto implementado com a CNN na pista real. Apesar do modelo deste condutor não ter sido treinado com imagens de provas em ambiente real, este condutor mostrou ser capaz de conduzir o veículo na pista real, mesmo com ruído, como mostra a figura 4.7.

⁷<https://youtu.be/bYYg-m-N2o4>. Acedido em 16/05/2018

⁸<https://youtu.be/WLpc6mLg9fg>. Acedido em 16/05/2018



Figura 4.7: Imagem de entrada da ANN no mundo real

Os exemplos do *dataset* de treino foram obtidos a uma velocidade linear constante de 0.8 m/s no ambiente de simulação e testados no mundo real a 0.4 m/s, pelo que a saída da rede apenas é válida para essa velocidade. Se se quiser usá-la para velocidades diferentes, é necessário ajustar a saída em conformidade. A relação entre a velocidade angular e a velocidade linear é dada pela equação 4.1, em que c representa a curvatura e ω_0 representa a velocidade angular correspondente à velocidade linear de treino v_0 . Assim, para uma velocidade linear v , a velocidade angular a aplicar ao veículo é dada pela equação 4.2.

$$\omega_0 = v_0 c \quad (4.1)$$

$$\omega = \frac{v}{v_0} \omega_0 \quad (4.2)$$

Aplicando esta equação, foi possível conduzir, na pista real, a 2.5 metros por segundo.

O vídeo na hiperligação ⁹ mostra parte de uma prova do ROTA com o condutor com SL na prova D1. Na classificação geral, o ROTA ficou em 3º lugar (apêndice A).

4.2 PILOTO IMPLEMENTADO COM APRENDIZAGEM POR REFORÇO

O piloto implementado com SL, utilizou para o treino um *dataset*, construído por um humano. Assim, o seu desempenho será, no máximo, igual ao desempenho de quem criou o *dataset*. Para tentar ultrapassar essa limitação, o agente poderá aprender sozinho a conduzir o veículo, utilizando técnicas de RL.

Esta secção descreve o processo de implementação do piloto utilizando RL. Num primeiro passo, foi necessário criar o controlador do ambiente de simulação, que servirá de interface entre o algoritmo a ser treinado e o ambiente de simulação (secção 4.2.1).

Foram escolhidos dois algoritmos de RL, o DDQN e o DDPG. Escolheu-se o algoritmo DDQN pois é um derivado no algoritmo DQN, simples de implementar e muito usado na comunidade de RL para comparação de algoritmos. O algoritmo DDQN reduz o sobreotimismo

⁹<https://youtu.be/kWZuz1CCZoA>. Acedido em 16/05/2018

do algoritmo DQN. Escolheu-se também o algoritmo DDPG, visto ser desenhado para problemas com espaço de estados e espaço de ação contínuos.

Escolhidos os algoritmos a utilizar, desenhou-se a arquitetura dos modelos das ANNs utilizados pelos algoritmos (secção 4.2.3) e escolheram-se os hiperparâmetros a utilizar (secção 4.2.4). Feitas as escolhas da arquitetura e dos hiperparâmetros, pode-se implementar e treinar o algoritmos (secção 4.2.5). Finalmente, são mostrados e comparados os resultados obtidos para os dois algoritmos (secção 4.2.6).

O código criado para implementar o agente com DDQN foi adaptado de ¹⁰ e para implementar o agente com DDPG foi adaptado o código de ¹¹. Este código foi adaptado de maneira a implementar as arquiteturas descritas na secção 4.2.3, os hiperparâmetros descritos na secção 4.2.4 e a aprendizagem descrita na secção 4.2.5. O código para implementar o ruído de Ornstein-Uhlenbeck foi retirado de ¹².

4.2.1 Controlador do Ambiente de Simulação

Para treinar o agente utilizando RL é necessário um controlador do ambiente capaz de avaliar o estado atual, receber ações do agente a treinar e enviá-las para o ambiente, calcular uma recompensa por determinada ação, verificar se um determinado episódio terminou e reiniciar o ambiente se isso se verificar.

O controlador criado para o ambiente ROTA é inspirado em [43], com base nos controladores de ambiente do Gym¹³ da OpenAI. Este controlador de ambiente é capaz de avaliar o estado do ambiente de simulação no Gazebo e enviar comandos para o veículo simulado.

A figura 4.8 representa a arquitetura do controlador do ambiente. A interface entre o algoritmo e o ROS é semelhante à dos controladores do Gym, facilitando a transposição de algoritmos criados que utilizam esses controladores de ambiente para este. Esta interface é responsável por traduzir as chamadas dos seus métodos em mensagens e chamadas de serviços ROS, que controla o simulador Gazebo.

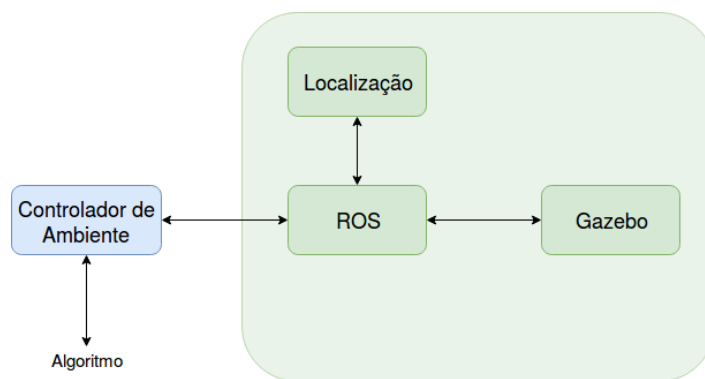


Figura 4.8: Arquitetura do Controlador de Ambiente

¹⁰<https://medium.com/@gtnjuvin/my-journey-into-deep-q-learning-with-keras-and-gym-3e779cc12762>. Acedido em 02/05/2018

¹¹<https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html>. Acedido em 13/03/2018

¹²<https://github.com/openai/baselines>. Acedido em 02/05/2018

¹³<https://gym.openai.com/>. Acedido em 16/05/2018

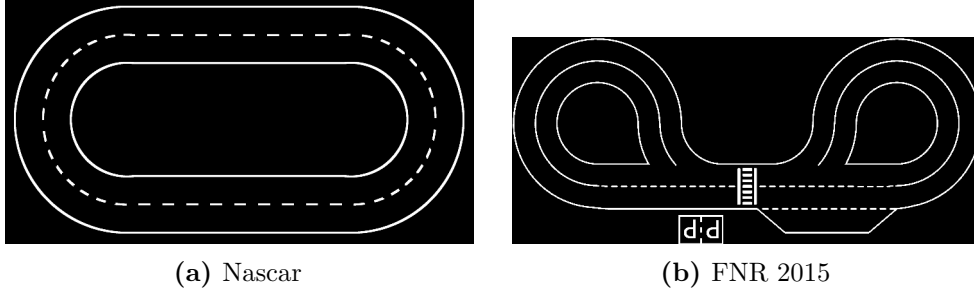


Figura 4.9: Pistas usadas na simulação

Esta interface é capaz de enviar para o algoritmo a treinar dois tipos de valores de entrada: a imagem recebida pela câmara; ou valores de baixa dimensão, como a orientação e a localização transversal à via. Estes dois valores são retirados do localizador do ROTA [10].

Quando o localizador não consegue obter uma estimativa confiável, a orientação do veículo for maior que um determinado valor ou quando o veículo sair de da pista, o controlador informa o algoritmo do fim do episódio.

A figura 4.9 mostra as pistas utilizadas na simulação. No entanto, verificou-se que o algoritmo de localização é instável, principalmente na pista da figura 4.9b, indicando em alguns casos a posição transversal errada, ou seja, a via errada. Logo, decidiu-se usar só a pista da figura 4.9a.

Tipos de Transição de Passos

Para cada passo no simulador, experimentou-se duas formas de transição, bloqueante e não bloqueante. Na forma bloqueante, implementado pelo controlador de ambiente de [43], quando se efetua um passo na simulação, a interface continua a simulação, envia o comando, recebe a informação sensorial e pausa a simulação. Na forma não bloqueante, a interface não pausa a simulação durante os passos. Este tipo de transição de passo é semelhante ao treino no mundo real, no entanto, são ignorados os intervalos de tempo entre as chamadas do método que simulam cada passo. Todavia, durante alguns testes, quando a carga de computação era elevada, verificou-se que a chamada dos serviços de pausa poderiam ocorrer num tempo variável, tempo esse ignorado para o treino do agente e levando a inconsistências nesse treino.

Cálculo da Recompensa

Para o cálculo da recompensa utilizou-se a equação 4.3, onde R_t é a recompensa no momento t , θ_t é a orientação do veículo, em radianos, relativamente ao eixo da pista no momento t , x_{ref} é a posição transversal de referência em relação ao centro da via e x_t é a posição transversal do veículo no momento t . x_{ref} e x_t estão em metros.

$$R_t = \cos(\theta_t) - \sin(|\theta_t|) - 1.5 * |x_{ref} - x_t| \quad (4.3)$$

O objetivo dessa equação é recompensar orientações próximas de uma paralela à via com $\cos(\theta_t)$, penalizar orientações próximas de uma perpendicular à via com $-\sin(|\theta_t|)$ e penalizar

quanto mais afastado estiver o veículo do centro da via com $-|x_{ref} - x_t|$. Quando o localizador não consegue localizar dentro da pista, a recompensa é igual a -2 e o episódio termina.

Também se utilizou a equação 4.4.

$$R_t = \cos(\theta_t) - \min(2 \cdot |x_{ref} - x_t| + 0.15, 20 \cdot |x_{ref} - x_t|^2) \quad (4.4)$$

Nesta equação, eliminou-se a penalização de orientações próximas de uma perpendicular à via e penalizou-se mais o afastamento do veículo do centro da via.

Para ambas as equações, a orientação encontra-se no intervalo $[-\pi/2, \pi/2]$.

4.2.2 Representação dos Espaços dos Estados e das Ações

Para além da imagem como representação dos estados, como foi utilizado no piloto implementado com SL, existem também as entradas de baixa dimensão, a orientação paralela à faixa e a posição transversal à faixa, recolhidas a partir dos histogramas construídos em [10]. Este tipo de representação é utilizada pelo Piloto original. Decidiu-se utilizar as entradas de baixa dimensão pois são necessários muito mais episódios para obter uma política quase ótima, em RL, e consequentemente muito mais tempo, do que passagens pelo *dataset* para obter um modelo quase ótimo em SL. Isto deve-se ao facto de que no treino de algoritmos de RL, estes têm de perceber por si só, através de tentativa e erro, quais são as melhores ações, enquanto que no treino de algoritmos de SL é utilizado um *dataset* com exemplos contendo as ações a tomar para cada estado.

Em relação à ação escolhida pelo agente, decidiu-se usar a correção da curvatura, ou seja, a ação escolhida pelo agente é adicionada à curvatura anterior, sendo que a curvatura inicial é igual a 0.0 m^{-1} . Uma das razões para esta escolha foi a generalização da ação para diferentes velocidades lineares, ao contrário da velocidade angular, utilizada na implementação do piloto com SL, que só é válida para a velocidade linear de treino. Outra das razões é que o espaço de correções de curvatura poderá ser menor que o espaço de curvaturas possíveis, útil para o DDQN. Para utilizar este espaço de ações, é necessário adicionar ao espaço de estados a curvatura do passo anterior. O ROTA permite curvaturas no intervalo $[-1; 1]$.

4.2.3 Arquitetura dos Modelos

Nesta secção são descritos os modelos utilizados pelos algoritmos DDQN e DDPG.

Double Deep Q Network

O modelo utilizado para o algoritmo DDQN é uma ANN com duas camadas intermédias totalmente ligadas. A figura 4.10 representa esta arquitetura, onde as suas camadas estão especificadas na tabela 4.2.

Como o algoritmo DDQN é utilizado para espaços de ações discretos, foi necessário discretizar o espaço de ações (correção de curvatura). Assim, dividiu-se este espaço em unidades com diferença entre si de 0.1, resultando em 21 valores. As saídas da rede correspondem ao resultado da função ação-valor para cada ação. Por isso, a função de ativação desta camada é uma função linear.

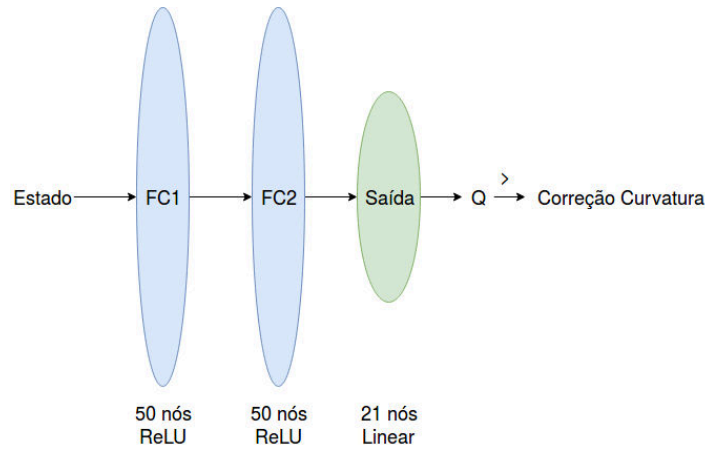


Figura 4.10: Arquitetura da ANN do piloto implementado com DDQN

Tabela 4.2: Arquitetura da rede neuronal do piloto implementado com DDQN

Nome da Camada	N de nós	Função de ativação
Entrada	3	-
FC1	50	ReLU
FC2	50	ReLU
Saída	21	Linear

Deep Deterministic Policy Gradient

Como descrito na secção 3.2.5, o algoritmo utiliza duas arquiteturas de modelos, uma para os modelos do ator e a outra para os modelos do crítico. A arquitetura do modelo do ator está representada na figura 4.11 e a especificação das suas camadas na tabela 4.3.

A função de ativação da camada de saída, tal como na arquitetura da CNN do piloto com SL (secção 4.1.3), é uma tangente hiperbólica que limita a saída (a ação) entre -1 e 1.

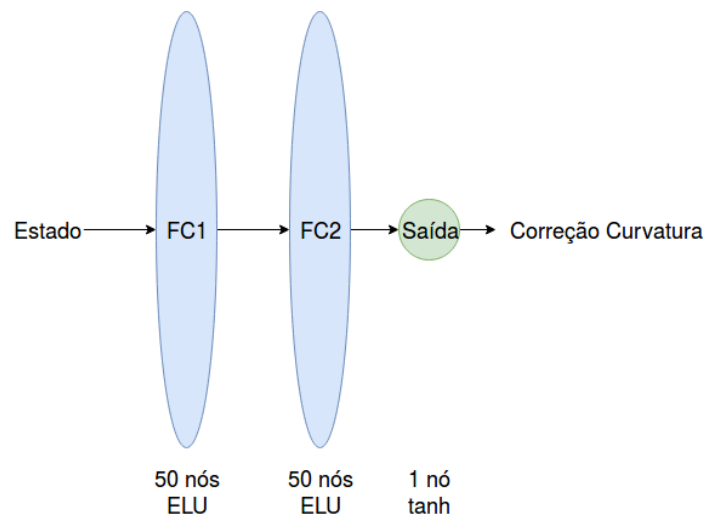


Figura 4.11: Arquitetura da ANN da função ator do piloto implementado com DDPG

Tabela 4.3: Arquitetura da rede neuronal da função ator do piloto implementado com DDPG

Nome da Camada	Nº de nós	Função de ativação	α
Estado	3	-	-
FC1	50	ELU	1
FC2	50	ELU	1
Saída	1	tanh	-

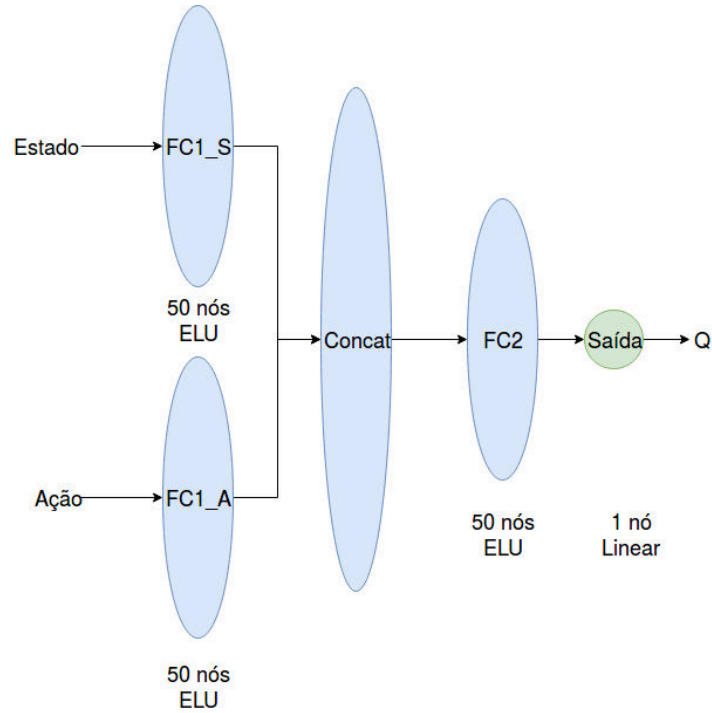


Figura 4.12: Arquitetura da ANN da função ator do piloto implementado com DDPG

A arquitetura dos modelos da função crítico está representada na função 4.12 e a especificação das suas camadas na tabela 4.4.

Este modelo tem duas entradas, uma para a ação escolhida pelo modelo do ator e outra para o estado. Estas duas entradas alimentam duas camadas que são concatenadas numa só. A saída da rede, como no modelo usado no DDQN (secção 4.2.3), representa o resultado da função ação-valor, e por isso a função de ativação é uma função linear.

Tabela 4.4: Arquitetura da rede neuronal da função crítico do piloto impleentado com DDPG

Nome da Camada	Nº de nós	Função de ativação	α
Estado	3	-	-
Ação	1	-	-
FC1_S	50	ELU	1
FC1_A	50	ELU	1
Concat	-	-	-
FC2	50	ELU	1
Saída	1	Linear	-

4.2.4 Hiperparâmetros

Nesta secção são descritos os hiperparâmetros utilizados na aprendizagem dos agentes para algoritmos DDQN e DDPG.

Double Deep Q Network

Para treinar o agente implementado com DDQN, utilizaram-se os hiperparâmetros descritos na tabela 4.5.

Tabela 4.5: Hiperparâmetros de treino de DDQN

Função de custo	MSE
Otimizador	Adam
Taxa de aprendizagem	0.0001
γ	0.99
τ	0.01
Taxa de atualização de ϵ por episódio	1/2500
Capacidade da memória	25000
Exemplos do conjunto de treino	32

Deep Deterministic Policy Gradient

Para treinar o agente implementado com DDPG, utilizaram-se os hiperparâmetros descritos na tabela 4.6.

Tabela 4.6: Hiperparâmetros de treino de DDQN

Função de custo	MSE
Otimizador	Adam
Taxa de aprendizagem	
Ator	0.00005
Crítico	0.0005
γ	0.9
τ	0.001
Taxa de atualização de ϵ por episódio	1/2500
Capacidade da memória	50000
Exemplos do conjunto de treino	128
Ruído Ornstein-Uhlenbeck	
θ	0.15
σ	0.2

4.2.5 Aprendizagem

O treino dos algoritmos, cada episódio pode-se dividir em três partes. Na primeira parte, o algoritmo age em cada passo até o episódio terminar, recolhendo informação para a fase seguinte. Na segunda fase, quando o episódio termina, o algoritmo treina o(s) modelo(s). A última parte, a validação, serve para validar a aprendizagem do algoritmo. O treino termina quando se observar que os algoritmos convergiram ou divergiram.

Esta secção descreve mais detalhadamente o treino dos algoritmos DDQN e DDPG.

Double Deep Q Network

Na fase de exploração, o algoritmo recebe o estado inicial do episódio. Com este estado, é calculada a ação a tomar, ou através da rede ou aleatoriamente, com o método de exploração ϵ -greedy. Neste método de exploração, o agente toma ações aleatórias com uma probabilidade ϵ , que diminui ao longo dos episódios, quando o agente começa a conhecer como agir. A ação é enviada para o controlador do ambiente, que efetua a ação, observa o novo estado, calcula a recompensa a atribuir e verifica se o episódio terminou. Esta informação é devolvida ao algoritmo de aprendizagem, que a guarda numa memória, juntamente com o estado atual. Estes passos são repetidos até o episódio terminar.

Findo o episódio, na fase de treino, o algoritmo retira da memória um conjunto de resultados dos passos, calcula os *Q Values* para esses passos e treina a rede. No fim, a rede *target* e a taxa de exploração são atualizadas. Para atualizar a rede *target*, utilizou-se a técnica do algoritmo DDPG para atualizar a rede *target*, visto que esta técnica atualiza esta rede de forma gradual em todos os episódios, em vez de atualizar em alguns episódios de forma abrupta.

A última fase, a validação, ocorre em alguns episódios, não ocorrendo nestes as outras duas fases. Esta fase é semelhante à primeira, exceto pela adição do ruído na ação e a salvaguarda da informação. As recompensas ganhas são guardadas para análise da evolução da aprendizagem do algoritmo.

Deep Deterministic Policy Gradient

Na fase de exploração, o algoritmo recebe o primeiro estado do episódio. De seguida, prevê a ação segundo este estado com a rede ator e é adicionado ruído Ornstein-Uhlenbeck [35], para explorar outras ações que não as previstas por esta rede. A ação é enviada para o controlador do ambiente, que efetua a ação, observa o novo estado, calcula a recompensa a atribuir e verifica se o episódio terminou. Esta informação é devolvida ao algoritmo de aprendizagem, que a guarda numa memória, juntamente com o estado atual. Estes passos são repetidos até o episódio terminar.

Findo o episódio, na fase de treino, o algoritmo retira da memória um conjunto de resultados dos passos, calcula os *Q Values* para esses passos com a rede do crítico e treina esta rede. De seguida, a rede do ator é treinada e as redes *target* e a taxa de exploração são atualizadas.

A última fase, a validação, ocorre em alguns episódios, não ocorrendo nestes as outras duas fases. Esta fase é semelhante à primeira, exceto pela adição do ruído na ação e a salvaguarda da informação. As recompensas ganhas são guardadas para análise da evolução da aprendizagem do algoritmo.

4.2.6 Resultados

Double Deep Q Network

Ao longo da avaliação e do treino, obtiveram-se valores da soma das recompensas obtidas por episódio de avaliação, utilizando a equação 4.3, e a média da função custo por episódio de treino. Estes valores estão representados nos gráficos das figuras 4.13 e 4.14, cujas linhas mais opacas representam valores suavizados e as linhas mais transparentes representam os valores recolhidos.

No primeiro gráfico, observa-se que a soma das recompensas por episódio de avaliação diminui até ao episódio 680. Observou-se que durante este período, o piloto tentava conduzir orientado pela faixa, mas afastado da posição ideal. A partir deste momento, o piloto tentou conduzir mais próximo da posição ideal e orientado com a faixa, aumentando as recompensas recebidas.

Observa-se que o gráfico dos valores recolhidos contém uma grande variância. Um dos fatores observados foi o término antecipado do episódio por parte do controlador do ambiente de simulação. Este término antecipado deve-se ao facto de que o localizador da posição transversal à faixa de rodagem indicar erradamente a posição do veículo, localizando-o fora da faixa de rodagem, enquanto que o veículo se encontrava dentro desta.

No segundo gráfico, pode-se observar que a média da função custo por episódio aumentou até ao episódio 300, afastando os valores da função ação-valor calculado pelo modelo dos

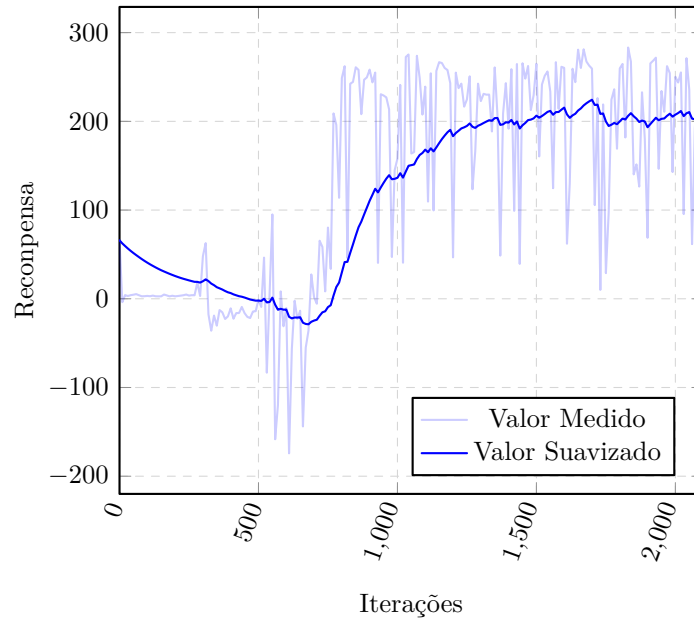


Figura 4.13: DDQN: Evolução da Recompensa nos Episódios de Avaliação. A suavização corresponde a uma soma entre 0.95 do valor anterior suavizado e 0.05 do valor atual medido.

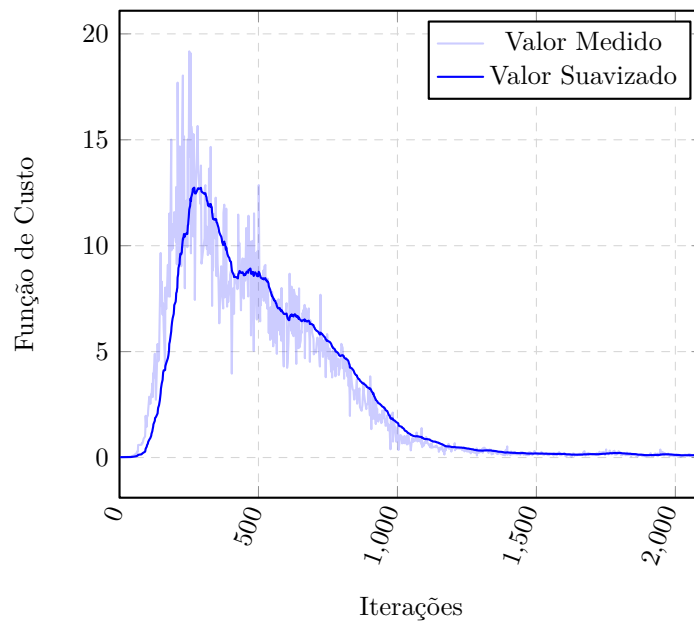


Figura 4.14: DDQN: Evolução da Função de Custo nos Episódios de Treino. A suavização corresponde a uma soma entre 0.95 do valor anterior suavizado e 0.05 do valor atual medido.

valores esperados. A partir deste episódio, o valor da função custo diminui para valores próximos de 0, o que significa que o modelo se ajustou às recompensas recolhidas.

O vídeo da hiperligação ¹⁴ mostra o piloto implementado com DDQN a conduzir na pista utilizada para treino. Como se pode ver neste vídeo, o veículo ainda oscila, para encontrar a melhor orientação e melhor posição. Uma das razões destas oscilações ocorrerem é o facto de

¹⁴<https://youtu.be/0vTXUWBQJvc>. Acedido em 23/05/2018

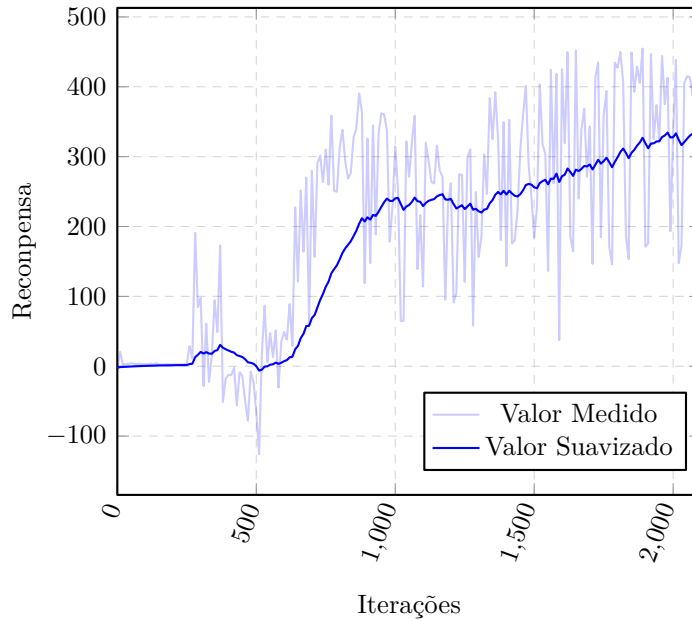


Figura 4.15: DDQN utilizando a segunda função de custo: Evolução da Recompensa nos Episódios de Avaliação. A suavização corresponde a uma soma entre 0.95 do valor anterior suavizado e 0.05 do valor atual medido.

ter sido necessário discretizar o espaço das ações. Para além das oscilações, o veículo também sai de pista.

Após a análise destes resultados, experimentou-se a função de recompensas 4.4, obtendo-se os gráficos da figura 4.15 para a soma das recompensas obtidas por episódio de avaliação e os gráficos da figura 4.16 para a média da função custo por episódio de treino.

Ao analisar os gráficos da figura 4.15, observa-se que a soma das recompensas aumenta até ao episódio 400, e, a partir deste episódio, diminui até ao episódio 500. Este comportamento é semelhante aos gráficos da figura 4.13, em que o piloto se preocupa mais em aproximar a orientação em relação ao eixo da pista à orientação ideal, em vez de aproximar a posição transversal ao centro da pista da posição ideal. As somas das recompensas aumentam até ao episódio 950, onde estabiliza até ao episódio 1200. Ao observar os gráficos da figura 4.16, é neste intervalo que a média da função custo começa a diminuir. A partir do episódio 1200, a soma das recompensas começa a aumentar, e a média da função custo estabiliza. Observando os gráficos dos valores reais das figuras 4.15 e 4.16, constata-se que contém uma variância muito grande, pelas mesmas razões que existem nos gráficos correspondentes à experiência anterior.

O vídeo da hiperligação ¹⁵ mostra o piloto implementado com DDQN treinado com a nova função de recompensas a conduzir na pista utilizada para treino. Comparando com o vídeo anterior, este piloto mantém-se na via correta por mais tempo, devido à remoção da penalização de orientações próximas de uma perpendicular à via e à maior penalização pelos desvios da posição transversal em relação à posição ideal. Por fim, o vídeo da hiperligação ¹⁶

¹⁵<https://youtu.be/H04K16G7eIY>. Acedido em 29/05/2018

¹⁶<https://youtu.be/PrWvCvb9Eig>. Acedido em 29/05/2018

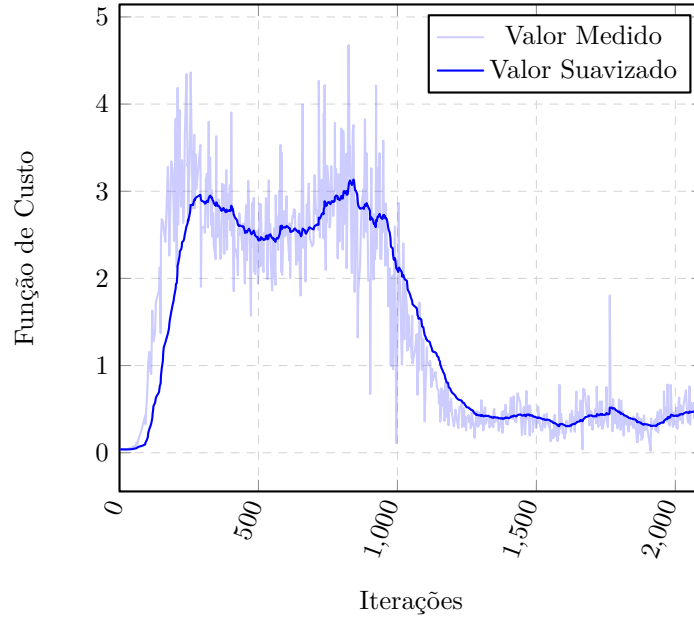


Figura 4.16: DDQN utilizando a segunda função de custo: Evolução da Função de Custo nos Episódios de Treino. A suavização corresponde a uma soma entre 0.95 do valor anterior suavizado e 0.05 do valor atual medido.

mostra este piloto a conduzir na pista simulada da prova de Condução Autônoma do FNR de 2018. Observando este vídeo, constata-se que o veículo sai de pista, principalmente nas curvas para a direita, inexistentes na pista de treino, conseguindo, no entanto, corrigir a sua posição.

Double Deep Policy Gradient

Em relação ao algoritmo DDPG, os resultados obtidos estão representados nas figuras 4.17 e 4.18, que representam, respetivamente, a soma das recompensas por episódio, na fase de avaliação, e a média por episódio da função custo da ANN do crítico, na fase de treino. O treino foi efetuado usando como função de recompensa a dada pela equação 4.4.

No gráfico da figura 4.17, observa-se que a soma das recompensas por episódio aumenta a partir do episódio 70 até ao episódio 150. O máximo da soma das recompensas verifica-se no episódio 220. Depois do episódio 230, esta soma decresce, tornando-se aproximadamente constante a partir do episódio 300, ou seja, o algoritmo divergiu. Observando o gráfico da figura 4.18, a média da função custo por episódio decresce até ao episódio 30, a partir do qual não sofre grandes alterações até ao episódio 60. A partir deste episódio, até ao episódio 175, a média da função custo aumenta, diminuindo até ao episódio 225. Analisando estes resultados, o algoritmo DDPG divergiu.

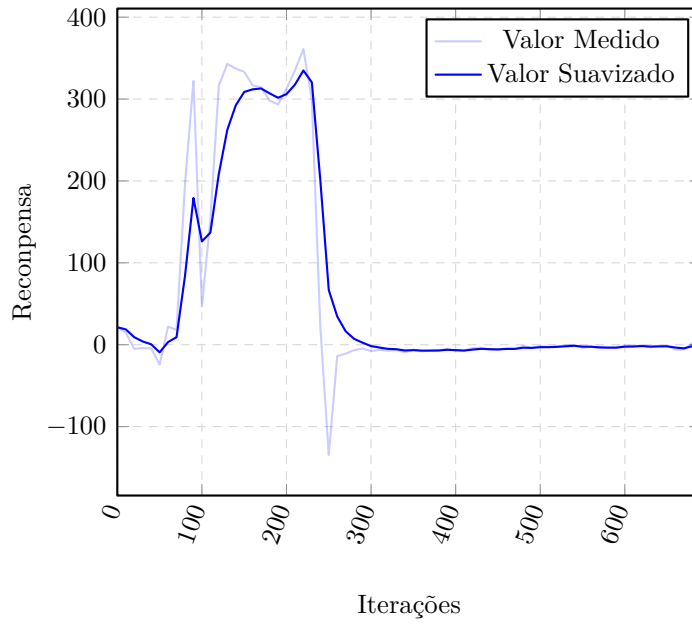


Figura 4.17: DDPG: Evolução da Recompensa nos Episódios de Avaliação. A suavização corresponde a uma soma entre 0.6 do valor anterior suavizado e 0.4 do valor atual medido.

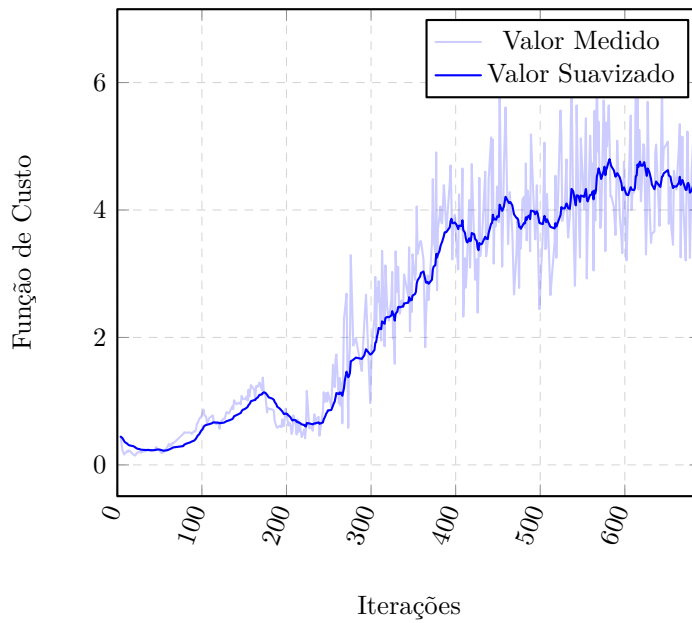


Figura 4.18: DDPG: Evolução da Função de Custo nos Episódios de Treino. A suavização corresponde a uma soma entre 0.95 do valor anterior suavizado e 0.05 do valor atual medido.

Comparação de resultados

Comparando os resultados obtidos pela utilização das duas funções de recompensa no algoritmo DDQN, e apesar do algoritmo ter convergido nos dois casos, obtiveram-se melhores resultados com a função de recompensa da equação 4.4 do que com a função de recompensa da equação 4.3, visto que o veículo percorreu uma distância maior na pista do ambiente de simulação utilizando a função da equação 4.4 do que utilizando a função da equação 4.3.

Comparando os resultados obtidos pelos algoritmos DDQN e DDPG, conclui-se que se obtiveram melhores resultados com o algoritmo DDQN, apesar de ter sido necessário discretizar o espaço de ações, visto que o primeiro algoritmo convergiu, ao contrário do segundo.

Conclusão

No contexto desta dissertação, pretendia-se desenvolver o módulo de condução (piloto) do veículo ROTA, aplicando Aprendizagem Automática. Para concretizar este objetivo, seguiram-se duas abordagens, uma utilizando SL e outra utilizando RL.

Na primeira abordagem, o módulo foi implementado utilizando uma CNN, tendo como entrada uma versão pré-processada da imagem da estrada captada pela câmara acoplada ao veículo, devolvendo a velocidade angular a aplicar. O *dataset*, utilizado no treino desta CNN, foi construído através de dados recolhidos e processados da condução manual do veículo, utilizando um comando remoto, na pista do ambiente de simulação, já existente neste projeto. A rede treinada obteve um bom desempenho, não só na pista simulada, como também na pista real, durante o FNR de 2018, que decorreu em Torres Vedras. É de realçar que, embora a CNN tenha sido treinada com uma velocidade constante de 0,4 m/s, conseguiu conduzir corretamente o veículo a velocidades superiores, com um máximo de 2,5 m/s. No entanto, foi necessário ajustar o valor devolvido pela rede (velocidade angular) à velocidade linear a que o veículo percorria a pista, porque este valor só é válido para a velocidade linear de treino. Para tal, foi necessário multiplicar o valor devolvido pela rede pela razão entre a velocidade pretendida e a usada no treino.

Podem-se retirar duas conclusões a partir dos resultados obtidos. A primeira, é que o ambiente de simulação é bastante realista na representação do veículo e da pista. A segunda, é que a utilização de simulação é adequada para o treino de ANNs que venham a ser utilizadas em ambiente real. Convém realçar que o primeiro contacto do veículo, utilizando o módulo de condução implementado com SL, com uma pista real decorreu durante o dito Festival, uma vez que não existe nenhuma pista no laboratório onde este trabalho foi desenvolvido.

A maior parte do desenvolvimento do módulo de condução com RL aconteceu após o FNR, pelo que não houve oportunidade de testar este módulo em ambiente real. Nesta abordagem considerou-se que a entrada da rede seria a saída dada pelo módulo de localização na faixa de rodagem já existente e o valor de saída a correção a aplicar na curvatura anterior. Foram utilizados dois algoritmos diferentes para desenvolver o módulo de condução: o DDQN e

o DDPG. Para o seu desenvolvimento, foi necessário adaptar previamente o ambiente de simulação, de modo a permitir a obtenção de estados da simulação, a aplicação de ações e o cálculo de recompensas. Para cada uma das soluções, foram desenhadas as arquiteturas das redes neuronais, foram escolhidos os hiperparâmetros adequados e foram realizadas as operações de treino.

No fim do treino, compararam-se os resultados obtidos por estes dois algoritmos. Comparando os dois módulos de condução, o módulo de condução implementado com DDQN obteve melhores resultados do que o módulo de condução implementado com DDPG.

Ao analisar os resultados obtidos pelos módulos de condução implementados com SL e RL, foi possível criar um módulo de condução implementado com SL capaz de conduzir o ROTA nas pistas de treino e de teste no ambiente de simulação, e também capaz de conduzir numa pista em ambiente real. No entanto, o módulo de condução implementado com RL, foi capaz de conduzir na pista utilizada para treino, com algumas imperfeições, como oscilações e saídas de pista.

5.1 TRABALHO FUTURO

Os resultados obtidos durante o trabalho desta dissertação mostram que é possível construir um módulo de condução do *software* de controlo de alto nível do projeto ROTA com Aprendizagem Automática. É por isso credível a utilização da mesma abordagem para outros módulos, como sejam a deteção de obstáculos, a deteção do pórtico sobre a passadeira, ou a deteção dos cones das zonas de obras, utilizando as imagens da câmara ou a saída do LRF.

O módulo de localização mostrou-se instável na sua utilização não só para o cálculo das recompensas atribuídas às ações tomadas pelos agentes, como também na avaliação dos estados, estimando às vezes de forma errada a posição do veículo. Para remover a primeira dependência, o controlador teria que extrair do simulador a informação da localização do veículo e com ela calcular a distância à posição de referência e a orientação. Para remover a segunda dependência, os algoritmos podem utilizar as imagens captadas pela câmara RGB-D como estado, no entanto, como referido na secção 4.2.3, o processo de aprendizagem levaria muito mais tempo até que os modelos convirjam. Após a remoção destas dependências, poder-se-iam utilizar outras pistas para treinar e avaliar os algoritmos, nomeadamente as pistas utilizadas na prova de Condução Autónoma do FNR.

Também seria interessante experimentar novas funções de recompensa, o algoritmo DDPG e outros algoritmos de RL, com outras arquiteturas e com outros hiperparâmetros, para experimentar se convergiam com estes controladores.

Certificado da Classificação na Prova de Condução Autónoma

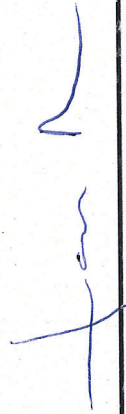
ROBOTICA 2018

XVIII RoboCup Portugal Open

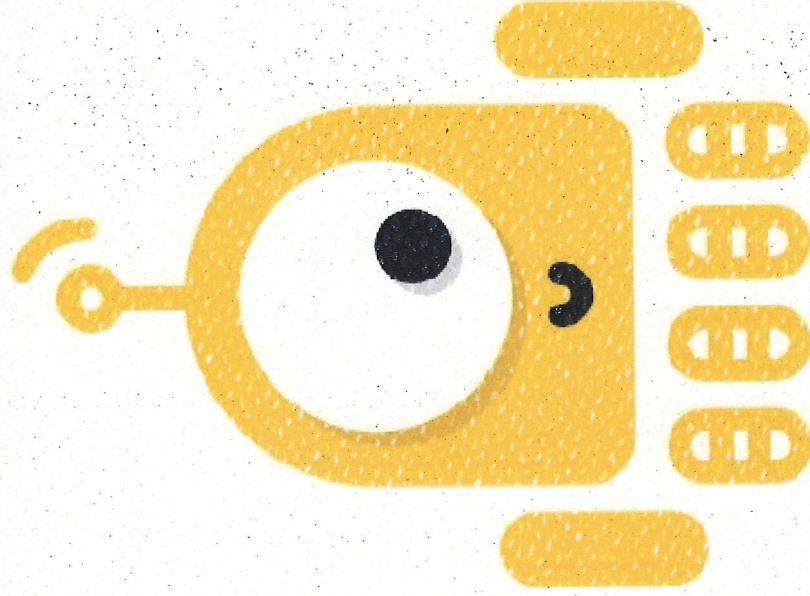
Certificado/ Certificate

Certifica-se que a equipa **Rota-2018-UA/IEETA**, da **UA**, obteve o **3º Lugar** na Prova de **Condução Autónoma**, no Festival Nacional de Robótica 2018, que decorreu de 25 a 29 de abril de 2018, em Torres Vedras.

We hereby certify that team **Rota-2018-UA/IEETA**, from **UA** was awarded **3rd prize** in the competition **Autonomous Driving** that took place from the 25th to the 29th of April 2018, in Torres Vedras.



Prof. Jaime Rei
General Chairman



Referências

- [1] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao e K. Zieba, «End to End Learning for Self-Driving Cars», *CoRR*, vol. abs/1604.07316, 2016. arXiv: 1604.07316. endereço: <http://arxiv.org/abs/1604.07316>.
- [2] V. Santos, J. Almeida, E. Ávila, D. Gameiro, M. Oliveira, R. Pascoal, R. Sabino e P. Stein, «ATLASCAR - Technologies for a computer assisted driving system on board a common automobile», em *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2010, pp. 1421–1427, ISBN: 9781424476572. DOI: 10.1109/ITSC.2010.5625031.
- [3] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard e L. D. Jackel, «Hand-written Digit Recognition with a Back-propagation Network», em *Proceedings of the 2Nd International Conference on Neural Information Processing Systems*, sér. NIPS'89, Cambridge, MA, USA: MIT Press, 1989, pp. 396–404. endereço: <http://dl.acm.org/citation.cfm?id=2969830.2969879>.
- [4] A. Krizhevsky, I. Sutskever e G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», em *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou e K. Q. Weinberger, eds., Curran Associates, Inc., 2012, pp. 1097–1105. endereço: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [5] K. Simonyan e A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. endereço: <http://arxiv.org/abs/1409.1556>.
- [6] Q. Le e T. Mikolov, «Distributed representations of sentences and documents», em *31st International Conference on Machine Learning, ICML 2014*, vol. 4, 2014, pp. 2931–2939, ISBN: 9781634393973.
- [7] J. Liu, W.-C. Chang, Y. Wu e Y. Yang, «Deep learning for extreme multi-label text classification», em *SIGIR 2017 - Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017, pp. 115–124, ISBN: 9781450350228. DOI: 10.1145/3077136.3080834.
- [8] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. Legresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan e Z. Zhu, «Deep speech 2: End-to-end speech recognition in English and Mandarin», em *33rd International Conference on Machine Learning, ICML 2016*, vol. 1, 2016, pp. 312–321, ISBN: 9781510829008.
- [9] L. B. Almeida, J. Azevedo, C. Carneira, P. Costa, P. Fonseca, P. Lima, A. F. Ribeiro e V. Santos, «Mobile robot competitions: Fostering advances in research, development and education in robotics», em *Portuguese Conference on Automatic Control (CONTROLO' 2000)*, 2000. endereço: <http://hdl.handle.net/1822/3296>.

- [10] J. B. G. Alves, «Navegação híbrida num veículo autónomo com direção Ackermann», tese de mestrado, Universidade de Aveiro, 2016. endereço: <http://hdl.handle.net/10773/18461>.
- [11] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler e A. Y. Ng, «ROS: an open-source Robot Operating System», em *ICRA Workshop on Open Source Software*, 2009.
- [12] W. S. McCulloch e W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *The bulletin of mathematical biophysics*, vol. 5, n° 4, pp. 115–133, dez. de 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259. endereço: <https://doi.org/10.1007/BF02478259>.
- [13] D. E. Rumelhart, G. E. Hinton e R. J. Williams, «Learning representations by back-propagating errors», *Nature*, vol. 323, n° 6088, pp. 533–536, 1986, ISSN: 00280836. DOI: 10.1038/323533a0. arXiv: [arXiv:1011.1669v3](https://arxiv.org/abs/1011.1669v3).
- [14] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. endereço: <http://neuralnetworksanddeeplearning.com/> (acedido em 28/02/2018).
- [15] S. Ruder, «An overview of gradient descent optimization algorithms», *CoRR*, pp. 1–14, 2016, ISSN: 0006341X. DOI: 10.1111/j.0006-341X.1999.00591.x. arXiv: 1609.04747. endereço: <http://arxiv.org/abs/1609.04747>.
- [16] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli e Y. Bengio, «Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization», NIPS'14, pp. 2933–2941, 2014. endereço: <http://dl.acm.org/citation.cfm?id=2969033.2969154>.
- [17] N. Qian, «On the momentum term in gradient descent learning algorithms», *Neural Networks*, vol. 12, n° 1, pp. 145–151, 1999, ISSN: 08936080. DOI: 10.1016/S0893-6080(98)00116-6.
- [18] J. Duchi, E. Hazan e Y. Singer, «Adaptive subgradient methods for online learning and stochastic optimization», *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011, ISSN: 15324435. DOI: 10.1109/CDC.2012.6426698. arXiv: [arXiv:1103.4296v1](https://arxiv.org/abs/1103.4296v1). endereço: <http://jmlr.org/papers/v12/duchi11a.html>.
- [19] M. D. Zeiler, «ADADELTA: an adaptive learning rate method», 2012, ISSN: 09252312. DOI: <http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503>. arXiv: 1212.5701. endereço: <http://arxiv.org/abs/1212.5701>.
- [20] D. P. Kingma e J. Ba, «Adam: a method for stochastic optimization», pp. 1–15, 2014, ISSN: 09252312. DOI: <http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503>. arXiv: 1412.6980. endereço: <http://arxiv.org/abs/1412.6980>.
- [21] X. Glorot, A. Bordes e Y. Bengio, «Deep Sparse Rectifier Neural Networks», em *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, 2011, pp. 315–323. endereço: <http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf>.
- [22] D.-A. Clevert, T. Unterthiner e S. Hochreiter, «Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)», *CoRR*, pp. 1–14, 2015, ISSN: 09226389. DOI: 10.3233/978-1-61499-672-9-1760. arXiv: 1511.07289. endereço: <http://arxiv.org/abs/1511.07289>.
- [23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever e R. R. Salakhutdinov, «Improving neural networks by preventing co-adaptation of feature detectors», 2012, ISSN: 9781467394673. DOI: [arXiv:1207.0580](https://arxiv.org/abs/1207.0580). endereço: <http://arxiv.org/abs/1207.0580>.
- [24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever e R. Salakhutdinov, «Dropout: a simple way to prevent neural networks from overfitting», *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014, ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807.
- [25] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel e U. Muller, «Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car», *CoRR*, vol. abs/1704.07911, 2017. arXiv: 1704.07911. endereço: <http://arxiv.org/abs/1704.07911>.
- [26] C. J. C. H. Watkins, «Learning from delayed rewards», tese de doutoramento, King's College, University of Cambridge, UK, 1989. endereço: http://www.cs.rhul.ac.uk/%7B~%7Dchrisw/new%7B%5C_%7Dthesis.pdf.

- [27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg e D. Hassabis, «Human-level control through deep reinforcement learning», *Nature*, vol. 518, n° 7540, pp. 529–533, 2015, ISSN: 14764687. DOI: 10.1038/nature14236. arXiv: 1312.5602.
- [28] M. Riedmiller, «Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method», em *Machine Learning: ECML 2005*, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge e L. Torgo, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328, ISBN: 978-3-540-31692-3.
- [29] S. Thrun e A. Schwartz, «Issues in using function approximation for reinforcement learning», *Proceedings of the 4th Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, pp. 1–9, 1993.
- [30] H. V. Hasselt, «Double Q-learning», em *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel e A. Culotta, eds., Curran Associates, Inc., 2010, pp. 2613–2621. endereço: <http://papers.nips.cc/paper/3964-double-q-learning.pdf>.
- [31] H. van Hasselt, «Insights in Reinforcement learning: formal analysis and empirical evaluation of temporal-difference learning algorithms», tese de doutoramento, 2011, pp. 1–282, ISBN: 9789039354964. endereço: http://homepages.cwi.nl/%7B~%7Dhasselt/insights%7B%5C_%7Dabstract.html.
- [32] H. v. Hasselt, A. Guez e D. Silver, «Deep Reinforcement Learning with Double Q-Learning», em *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, sér. AAAI’16, Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100. endereço: <http://dl.acm.org/citation.cfm?id=3016100.3016191>.
- [33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver e D. Wierstra, «Continuous control with deep reinforcement learning», 2015, ISSN: 1935-8237. DOI: 10.1561/22000000006. arXiv: 1509.02971. endereço: <http://arxiv.org/abs/1509.02971>.
- [34] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra e M. Riedmiller, «Deterministic policy gradient algorithms», *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014, ISSN: 1938-7228.
- [35] G. E. Uhlenbeck e L. S. Ornstein, «On the theory of the Brownian motion», *Physical Review*, vol. 36, n° 5, pp. 823–841, 1930, ISSN: 0031899X. DOI: 10.1103/PhysRev.36.823.
- [36] D. A. Pomerleau, *Neural Network Perception for Mobile Robot Guidance*. Norwell, MA, USA: Kluwer Academic Publishers, 1993, ISBN: 0792393732.
- [37] C. Chen, A. Seff, A. Kornhauser e J. Xiao, «DeepDriving: learning affordance for direct perception in autonomous driving», em *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, 2015, pp. 2722–2730, ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.312. arXiv: 1505.00256.
- [38] April Yu, Raphael Palefsky-Smith e Rishi Bedi, «Deep reinforcement learning for simulated autonomous vehicle control», rel. téc., 2016.
- [39] M. Vitelli e A. Nayeibi, «Carma: A deep reinforcement learning approach to autonomous driving», *CoRR*, 2016.
- [40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu e X. Zheng, «TensorFlow: large-scale machine learning on heterogeneous distributed systems», *Journal of Neuroscience*, 2016, ISSN: 0270-6474. DOI: 10.1038/nn.3331. arXiv: 1603.04467. endereço: <http://arxiv.org/abs/1603.04467>.
- [41] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang e A. Y. Ng, «Large scale distributed deep networks», *NIPS 2012: Neural Information Processing Systems*, pp. 1–11, 2012, ISSN: 10495258. DOI: 10.1109/ICDAR.2011.95. arXiv: fa.
- [42] L. Perez e J. Wang, «The effectiveness of data augmentation in image classification using deep learning», *CoRR*, 2017. arXiv: 1712.04621. endereço: <http://arxiv.org/abs/1712.04621>.

- [43] I. Zamora, N. G. Lopez, V. M. Vilches e A. H. Cordero, «Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo», *CoRR*, n^o August, pp. 1–6, 2016. arXiv: 1608.05742. endereço: <http://arxiv.org/abs/1608.05742>.